Université Catholique de Louvain
Faculté des Sciences Appliquées
Département d'Ingénierie Informatique

Securing Network Coordinate Systems

Promoteur: **Prof. Olivier Bonaventure**
Lecteurs: **Prof. Laurent Mathy**
Assistant Renaud De Landtsheer

Mémoire présenté en vue
de l'obtention du grade
d'ingénieur civil
en informatique
par
Damien Saucez

Louvain-la-Neuve
Année académique 2006–2007

to my grand father

Acknowledgements

First, I would like to thank Professor Olivier Bonaventure for giving me the possibility of working on such an interesting research topic.

I would like to thank Professor Laurent Mathy at Lancaster University for his suggestions.

I would thank the responsibles of the European-founded 034819 OneLab project which partially support this work.

I would also like to express my profound appreciation to Benoit Donnet at Université Catholique de Louvain. His help and suggestions helped me all along this project.

The comments concerning the concepts presented in this thesis provided by Pierre Gramme and Sébastien Mouthuy where helpful. I thank them for spending their time and competences.

Especially, I would like to give my special thanks to my parents and my brother for their support all along my studies.

Abstract

Many large-scale Internet applications optimize their overlay network to reduce latencies. Embedding coordinate systems like Vivaldi or NPS are valuable tools for this new range of applications since they propose light-weight algorithms that permit to estimate the latency between any pair of nodes without having to contact them first. It has been recently demonstrated that network coordinate systems in general are sensible to attacks. Typically, malicious nodes can lie about their coordinates and distort the coordinate space.

In this work, we propose a formal reputation model to detect misbehaving nodes. The reputation model rely on two new types of nodes, the RCA, a certification agent and the surveyors that monitors nodes. Based on the observation of surveyors, the RCA gives a reputation to each node. The reputation estimates the probability that the node is not malicious. In this work, we propose a new network coordinate system called RVivaldi. RVivaldi is an adaptation of Vivaldi that implements the reputation model.

Based on experiments on RVivaldi, we determine that coordinate systems with reputation are less sensible to attacks than the system without the reputation.

Résumé

Beaucoup d'applications de grande ampleur sur l'Internet optimisent leur topologie de sorte à réduire leur latence. Les systèmes de coordonnées comme Vivaldi et NPS sont des outils des plus intéressants pour ces applications. En effet, leurs algorithmes, légers, permettent d'estimer la latence entre n'importe quelle paire de noeuds sans avoir à les contacter d'abord. Hélas, il a récemment été montré que les systèmes de coordonnées sont sensibles aux attaques. Typiquement, les noeuds peuvent mentir au sujet de leurs coordonnées et déformer l'espace de coordonnées.

Dans ce travail, nous proposons un modèle formel de réputation qui permet de détecter les noeuds malveillants. Le modèle de réputation repose sur deux nouveaux types de noeuds. Le RCA, un agent de certification, et les surveillants qui monitor les noeuds. Sur base des observations faites par les surveillants, le RCA attribue une réputation à chaque noeud. La réputation estime la probabilité que le noeud ne soit pas malveillant. Dans ce travail, nous proposons aussi RVivaldi, un nouveau système de coordonnées. RVivaldi est un adaptation de Vivaldi qui implémente le modèle de réputation.

Sur base des expériences menées sur RVivaldi, nous avons pu déterminé que les système de coordonnées qui implémentent la réputation sont moins sensibles aux attaques que s'ils ne l'implémente pas.

Contents

List of Figures

List of Tables

Listings

1 Introduction

These last few years, many different application-level overlays (Chord [5], CAN [6]) have been proposed to support new range of applications from file sharing to Voice over IP (VoIP) and, more recently, IPTV. Most of these applications rely on the notion of proximity, usually defined as network delay or round-trip time (RTT), to discover close neighbours which ensure the optimality of the quality of the service built on the overlay.

Nevertheless, measuring latency on-demand is impractical due to its cost: One measurement per pair of nodes and the number of pairs is a quadratic function of the number of nodes. So that, the total number of measurements is $N*(N-1)$ for N nodes. Fig. 1 presents the quadratic evolution of the number of pairs of nodes (vertically) with the number of nodes (horizontally). Moreover, obtaining the information can exceed the cost of the effective process[1] [7, 8, 9].

It is important for the new applications presented above to limit the resources consumption to the maximum and particularly the number of on-demand measurements. *Network Coordinate Systems* (*NCS*) have been proposed to allow hosts to estimate delays without performing direct measurements and thus reduce the consumption of network resources [7, 8, 1, 9, 10, 3, 2, 11]. Every node of a coordinates-based approach computes its coordinates into a virtual geometric space such that the distance from itself to any host predicts the latency – called *distance* – to that node. Other metrics like bandwidth or the number of routers hops can be used for such systems. Unfortunately, while it is relatively easy to provide latency information, it is quite hard to provide good bandwidth estimation [12]. Indeed, bandwidth is sensitive to the exact path followed[2]. Hopefully, applications that need high-bandwidth can minimize latency as a first approximation since there is usually a rough correlation between low latency and high bandwidth. The revolution of the NCS is that it is no more needed to probe any arbitrary pair of nodes to estimate the distance between them. Nevertheless, NCS still need measurements but only to a constant number of well-chosen nodes. In addition, the NCS measurement layer can work in background and thus limit the time needed to choose the best neighbour when needed. Actually, if the coordinates of any two nodes are known, the estimated distance is computed by applying a distance function on these coordinates.

Coordinates-based approaches can be divided into two categories (*i*) the *Landmarks-based* services where some well-known nodes are trusted and used to compute coordinates of all other, and (*ii*) the *decentralized* services which is the opposite: Every node can act as a Landmark. The most well known technique of the last method is *Vivaldi* [2]. For the former we can cite *NPS* [3].

Content distribution and file sharing systems can benefit from network coordinates in order to select a number of replicated servers to fetch data from. *Azureus* [13], for instance, was the first large-scale real world application to use a coordinate system. Furthermore, the new "killer

[1] Measuring latency between a client and web servers can be more costly than accessing the page.

[2] A single low-bandwidth link dictates the bandwidth of the whole path.

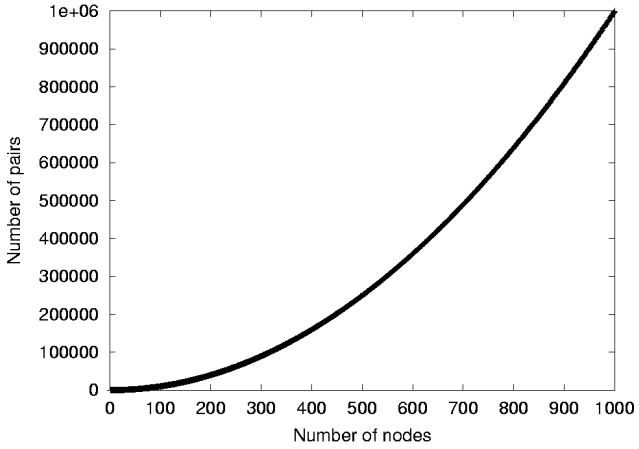


Figure 1.1: Evolution of the number of pairs of nodes with the number of nodes

application" *AllPeers* [14] that permits to share private content directly using a web browser also relies on NCS to take good decisions for the construction of the underlying topology.

Ledlie et al. [15] have shown that NCS are valuable tools for distributed systems relying on the topology of the network. However, due to their property of slow convergence, NCS must be deployed as always-on services available for higher level applications. But large-scale always-on services are prime target for attackers. Indeed, an attack might lead to a malfunction of many applications or overlays. A malicious node may alter coordinates or change the measurements themselves [4, 16, 17]. Indeed, coordinates are computed from information given by other nodes. If some nodes lie about their coordinates or latency, coordinates of the whole system can be altered [4, 16].

Attackers may also be interested by applications like Azureus or AllPeers since they are open-source and widely spread around the world. One can imagine modifying them to alter the coordinates space and disrupt the whole service or controlling all the traffic to achieve a *denial-of-service* (DoS) attack.

The attacks on coordinate systems can be divided into two different categories. The first one is performed when a honest node asks coordinates to a malicious one. The malicious node replies with false coordinates resulting in a bad latency prediction. Secondly, attackers disrupt the coordinates computation process itself resulting in a deformation of the space of both honest and malicious nodes (i.e. the predicted distances of the entire system are altered).

We propose a formal reputation model for securing coordinate systems. The key idea is to associate a *reputation* to every node. This reputation gives an information on the reliability of the node. The reputation is high for honest nodes and low for others. The reputation is based on how the nodes behaved in the past and how old are the nodes in the system. To compute the reputation, two new types of entities have to be added in the system: The *Reputation Computation Agent* (*RCA*), a certificating agent, and the *surveyors* that estimate the reputation of the classic nodes.

We also propose an adaptation of Vivaldi – *RVivaldi* – that implements the model. We evaluate RVivaldi by simulation using the King data [2] and [16]. We show that RVivaldi leads to a better accuracy of the coordinates than Vivaldi in presence of malicious nodes.

Finally, we would like to emphasize that RVivaldi has interested the research community. A paper, titled "A Reputation-Based Approach for Securing Vivaldi Embedding System", discussing the reputation model and a preliminary evaluation of RVivaldi has been accepted to appear in the EUNICE workshop. An extension of this paper is currently under progress. We aim at submitting it in early July.

This thesis is divided into five chapters. Chapter 2 surveys the most important NCS. This chapter describes the computation of the coordinates in the presented *embedding systems*. It also presents the differences between the different approaches and focuses on the two current "state-of-the art coordinate systems": Vivaldi and NPS.

Chapter 3 proposes a broad discussion of the security concerns with NCS based on the work of Kaafar et al. [4, 16]. In this chapter, we introduce the concepts of *external attacks* and *internal attacks*. In the first sort of attack, the malicious nodes do not participate in the coordinates computation but alter the exchanges between nodes in the coordinate system. On the contrary, in internal attacks, misbehaving nodes actively participate in the computation of the coordinates. We notice that it is harder to protect a system against internal attackers than external ones. This chapter, shows the necessity of proposing a formal security model for coordinate systems.

In Chapter 4 we present a global solution to solve these security issues. We first introduce the concepts of *experience*, *trust* and *reputation* and the theory of *uncertain probabilities*. After, we propose a *formal reputation model* that is a sort of framework that we adopt to secure coordinate systems. This formal model is based on the uncertain probabilities, the experience, the trust and the reputation. Finally, we propose RVivaldi, an adaptation of Vivaldi based on the reputation model. Our accepted paper presents this chapter [18].

Chapter 5 presents experimental results on RVivaldi. The experimental results are obtained by a simulation of four different types of attacks on RVivaldi . The results of the experiments show that RVivaldi is more tolerant than Vivaldi in the presence of malicious nodes. The results also highlight the best choice of the parameters introduced by the reputation model.

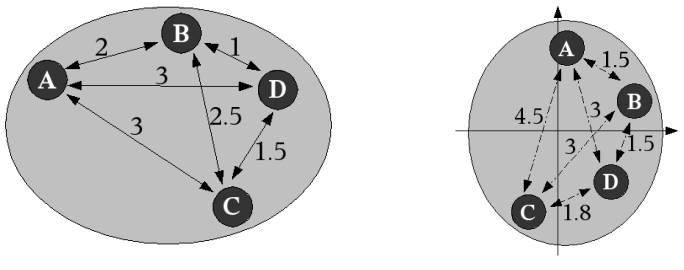
Finally, in Chapter 6 we summarize the main points detailed during the thesis and discuss remaining open issues.

2 Network Coordinate Systems

Network Coordinate Systems have been proposed to allow hosts to estimate delays between nodes without having to contact them first [7, 8, 1, 9, 10, 3, 2, 11]. Nodes compute their coordinates into a virtual geometric space such that the distance from itself to any host predicts the latency to that node. For instance, the distance $\|\vec{c}_1 - \vec{c}_2\|$ predicts the RTT between two nodes with coordinates $\vec{c}_1$ and $\vec{c}_2$ respectively [11]. The main idea of coordinates-based system is that every host maintains *coordinates*. The Internet is represented as a space where each point with a particular coordinates is a node. The more accurate the *coordinate space model*, the more accurate the prediction of the distances. For short, a coordinates-based system must propose a mapping of Internet hosts to a virtual space such that the distances in this last space are as close as possible to the actual distances between Internet hosts. In NCS, the coordinates are computed using the coordinates of a set of well-chosen nodes and the distance to them. This operation is performed in background to permit any upper-layer application to get optimal coordinates when needed. The correspondence between the real space and the virtual space is depicted in Fig. 2.1. In Fig. 2.1(a), the distances between four hosts on the Internet are represented. Fig. 2.1(b) presents the estimated distances in a virtual geometric space. The nodes are placed in a space such that their distances is a rough approximation of the reality. We can observe that the predicted distances are not perfect. This is one of the weakness of the coordinate systems but as we can see later, coordinate systems are still valuable tools.

Instead of using the distance, one can use metrics like the number of routers hops or bandwidth. Unfortunately some metrics are harder to use than others. For example, the bandwidth is sensitive to the exact path followed[1] and thus is quite hard to estimate [12].

[1] A single low-bandwidth link dictates the bandwidth of the whole path.



(a) Internet (b) Virtual space

Figure 2.1: Correspondence between the Internet and the virtual network

Hopefully, applications that need high-bandwidth can minimize latency as a first approximation since there is a rough correspondence between the two concepts.

There is two families of coordinate systems. First, in *landmarks-based* coordinate systems, a fixed set of well-known trusted nodes is used to compute coordinates of all other. Second, in *decentralized* coordinate systems, every node can be used for the computation of coordinates of any other.

Ng and Zhang propose an evaluation methodology for coordinate systems [8]. They propose a performance metric called *directional relative error*:

$$\frac{\text{predicted distance} - \text{measured distance}}{min(\text{measured distance}, \text{predicted distance})}. \tag{2.1}$$

The absolute value of directional relative error is called the *relative error* (Eqn. 2.2).

$$\frac{|\text{predicted distance} - \text{measured distance}|}{min(\text{measured distance}, \text{predicted distance})}. \tag{2.2}$$

Unless otherwise stated, d_{AB} is the measured distance between nodes A and B. The predicted distance is noted $\hat{d}_{AB}$.

A value of zero for Eqn. 2.1 implies a perfect prediction, a value of one implies the predicated distance is larger by a factor of two, and a negative one implies the predicted distance is smaller by a factor of two. This metric has been proposed to guard against the "always predict zero" policy that is impossible with a simple percentage error. The directional relative error is interesting to measure how well a predicted distance matches the corresponding measured distance. A negative value implies the predicted distance is too small and a positive value indicates the predicted distance is too large. However, the relative error is more relevant when considering general prediction accuracy. The relative error makes no difference between a x times bigger distance prediction and a x times smaller prediction.

Most of the coordinate systems map Internet hosts to a virtual Euclidean space to estimate distances. In such a space, the estimated distance $\hat{d}_{H_iH_j}$ from host H_i to node H_j is evaluated with a distance function defined as follow:

$$\hat{d}_{H_iH_j} = \sqrt{\sum_{l=1}^{d} \left(\vec{c}_{H_i} - \vec{c}_{H_j}\right)^2} \tag{2.3}$$

where $\vec{c}_{H_i}$ are the coordinates of H_i and $\vec{c}_{H_j}$ are the coordinates of H_j.

Moreover, Ng and Zhang have remarked that it is important to preserve the order of the predicted distances. Indeed, the purpose of a server-selection algorithm is to provide the server with the minimum latency, whatever the latency. The concept of *rank accuracy* has been introduced to estimate the preserving order.
After each experiment, a *measured ranked list* and a *predicated ranked list* are built. The nodes are ordered by measured distance in the measured ranked list. In the predicated ranked list, the nodes are ordered by their predicated distance. The ranked accuracy is defined has the

portion of paths correctly selected using the predicted ranked list (compared to the measured ranked list). A rank accuracy of 100% means a perfect predicated ranking. Gummadi et al. propose a methodology to evaluate the *preserving order* of distance estimation systems [19].

In the rest of this chapter, we survey the most important network coordinate systems and, finally, we compare them.

2.1 Survey

In this section, we survey different solutions proposed for the problem of distances prediction on Internet. First, we present *IDMaps* (Sec. 2.1.1), a precursor to Network Coordinate Systems. Second, we detail *GNP* (Sec. 2.1.2), the first ever Network Coordinate System. After, we talk about *Lighthouses* (Sec. 2.1.3), *BBS* (Sec. 2.1.4) and *PIC* (Sec. 2.1.5). Finally, we look at *Vivaldi* and its stable version *SVivaldi* (Sec. 2.1.6) and *NPS* (Sec. 2.1.7).

2.1.1 IDMaps

Even if IDMaps [7] – for Internet Distance Map Service – is not a network coordinate system and thus is not covered by this work, Francis et al. give the milestones for the problem of predicting network distances that lead to coordinate systems. One of the most important work in IDMaps is the observation that the three following key points must be followed to build a good distances prediction system:

1. Determine how the distance information is produced and which system produces it.
2. Determine how the distance information is transmitted from this producing system.
3. Determine the form of the information and how the distance information is used to produce the specific answers.

The NCS presented below in this work all follow theses points.

Francis et al. also propose to divide distance information into two categories. (*i*) *Load sensitive* which depends on network load and (*ii*) *raw* which are independent of the load. While load sensitive measurements change every time, raw measurements are on the order of hours if not days old. Raw distances only slowly reflect permanent topology changes and do not need frequent monitoring. On the contrary, load sensitive measurements require very frequent probes. Fig. 2.2 shows that if distances are not frequently measured, they may not reflect some network changes. On the contrary, if the distance is near reality, the number of measurements must be important. The frequency of the measurements is projected on the horizontal axis and the instantaneous accuracy is on the vertical axis. A small frequency means that only few measurements are required. An important instantaneous accuracy means that that the estimated distances always reflect the real distance. On the contrary, a low instantaneous accuracy means that the predicted distances might not be the perfect reflect of the reality.

Two distinct variations of the architecture of IDMaps are proposed: *Hop-by-hop* (*HbH*) and *End-to-End* (*E2E*). In both architectures, a new entity type is proposed: *Tracers*. The

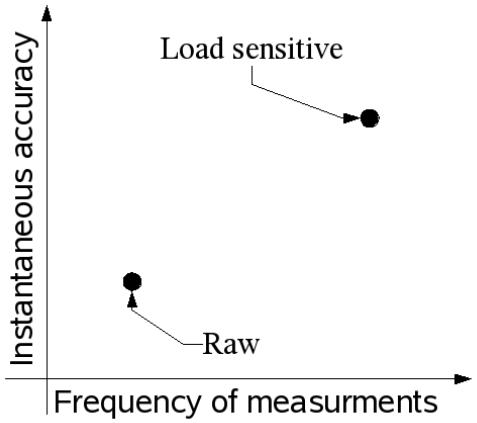


Figure 2.2: Comparison between raw distances and load sensitive distances

purpose of Tracers is to collect and advertise raw distances. The set of all these distances permit to build a *distance map* and clients can estimate distances between any two Internet hosts based on this distance map. In HbH the Tracers probe all transit backbone routers. On the contrary, only well-chosen nodes are probed in E2E.

In addition to Tracers, some *boxes* are distributed around the Internet. The position of boxes is such that every *Address Prefix (AP)*[2] is close to one or more boxes. A list of box-box distances and distances between the APs and their nearest boxes is maintained. Thus, the distance between any two APs can be estimated by the sum of the distances between APs and their nearest boxes and the distances between the boxes. The accuracy of the estimate depends on how close AP is to a box [7, 9].

The two following effects permit IDMaps not to have to know every distances between boxes:

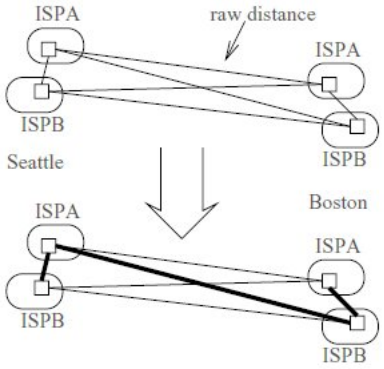
long-leg/short-leg effect : A given unknown distance can be estimated by concatenating a set of known distances together, provided that all but one of the known distances are short. If distance d_{AB} and d_{BC} are known, the triangle inequality bound the distance d_{AC} above by $d_{AB} + d_{BC}$, and below by $|d_{AB} - d_{BC}|$. Fig. 2.3(a) presents an illustration of this effect.

On the path effect : Sometimes a node is pretty much on the path of other nodes, thus the distance can be estimated by the concatenation of the distances between all nodes of the path. Fig. 2.3(b) illustrates this point.

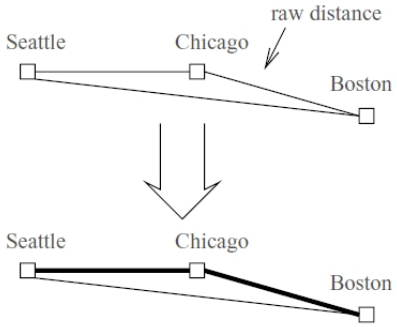
These two properties suppose that the routers always use low-latency paths (i.e., *efficient routing*).

Client of IDMaps estimates the distance from one IP address to another by taking the following three steps:

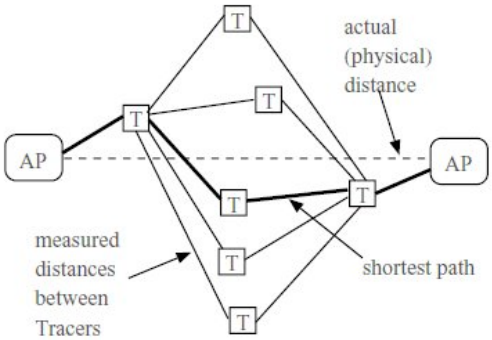
[2] An AP is a block of IP addresses that can be aggregated at a single *ISP* (*Internet Service Provider*) backbone router [7].



(a) "Long-leg/Short-Leg" effect (presented in [7])



(b) "On the Path" effect (presented in [7])



(c) Spanning tree (presented in [7])

Figure 2.3: The problem of redundant distances

1. Determine the AP of each address.
2. Determine which box the APs are connected to.
3. Run a *spanning-tree* algorithm over the topology of boxes of the source APs to the boxes of the destinations APs. The resulting distance is considered as the estimated distance.

The difference between HbH and E2E is the exact operation of the spanning-tree (step 3). A spanning-tree algorithm can be seen as a manipulation that transforms a connected graph into a tree where a path exists to every nodes [20]. Next two points present the operation for HbH and E2E:

In HbH model, the box-box topology is the actual Internet transit routers topology. Each backbone router is modeled as a box, and every physical link between them are box-box distances. The Tracers probe all transit backbone routers.

The operations performed in the spanning-tree (step 3) in HbH are:

1. Determine which AS each AP connects to.
2. Running a spanning tree algorithm over the inter-AS topology, calculate the AS path from one AS to the other.
3. For each AS in the AS path, starting with the source AS, calculate the shortest router path from the entry router (box) to the next AS (intra-AS topology).
4. Concatenate the calculated router paths to obtain the complete path.

In the E2E model the box-box distances are the distance measurements made between Tracers and the AP-box distances are these made from Tracers to AP. In this model, the boxes are nothing else than the Tracers. The spanning tree proposed in step 3 can be executed without additional mechanism.

Solving a *spanning-tree* over the topology of distances permit to solve the problem of determining if an intermediate box b is on the path between two others (see Fig. 2.3(c)).

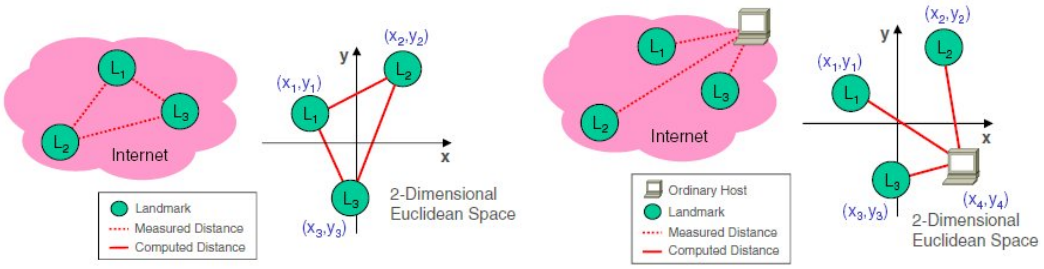
Both HbH and E2E use multicast groups to spread distance map through the network[3].

2.1.2 Global Network Positioning (GNP)

Ng and Zhang were the first to propose to use coordinates-based mechanisms to predict Internet network distances, proposing the *GNP* approach [8].

GNP is based on a two-part architecture. In a geometric space $\mathcal{S}$ of dimension d, a small set of at least $d+1$ well-known *landmarks* hosts compute their own coordinates. Any other host can thus computes its own coordinates based on the landmarks coordinates [1, 8].
In the following, $L_1, \ldots, L_N$ refer to the N landmarks nodes with $N \geq d+1$.

[3] Clients do not need to know Tracers and vice-versa



(a) Landmark operation (presented in [8])

(b) Ordinary host operation (presented in [8])

Figure 2.4: Operations in GNP

Landmark operations

The landmarks operation for a 2-dimensional Euclidean space is presented in Fig. 2.4(a).

The distance between landmarks nodes is the minimum of several measured ICMP ping messages (the raw RTT) for every landmark-to-landmark path. The distance between landmarks i and j, L_i and L_j is symmetric and denoted by $d_{L_iL_j}$.

With the measured distances, $d_{L_iL_j}, i > j$, a host computes the coordinates of the landmarks in the space. The goal of this computation is to find a set of landmarks coordinates $\vec{c}_{L_1}, \ldots, \vec{c}_{L_N}$ such that the overall error between the measured distances $d_{L_iL_j}$ and the computed distances $\hat{d}_{L_iL_j}$ is minimized. Formally, this host try to minimize the following objective function:

$$f_{obj1}\left(\vec{c}_{L_1}, \ldots, \vec{c}_{L_N}\right) = \sum_{L_i, L_j \in \{L_1, \ldots, L_N\} | i > j} Error\left(d_{L_iL_j}, \hat{d}_{L_iL_j}\right). \tag{2.4}$$

where $Error(\cdot)$ is an error measurement function which can be the following:

$$Error\left(d_{AB}, \hat{d}_{AB}\right) = \left(d_{AB} - \hat{d}_{AB}\right)^2. \tag{2.5}$$

Initially, landmarks coordinates are set at random but when a re-computation of landmarks coordinates is required, old coordinates are used to avoid large variation in coordinates.

We can see that every landmark measures its distances to other landmarks in Fig. 2.4(a). All these distances are used to optimize the position of the landmarks coordinates on the virtual space.

Ordinary hosts operations

The landmark's coordinates depends on the coordinates of all the other landmarks. Thus, a change in a landmark coordinates implies all the other landmarks to recompute their coordi-

nates. On the contrary, the coordinates of any "ordinary host" H is derived from coordinates of the landmarks and do not interfere with other node's coordinates.

To compute its own coordinates $\vec{c}_H$, host H measures the raw round-trip delays to the N landmarks. The computation of $\vec{c}_H$ is the result of the minimization of the overall error between the measured and the computed host-to-landmark distances. The function to minimize is the following:

$$f_{obj2}\left(\vec{c}_H^{\mathcal{S}}\right) = \sum_{L_i \in \{L_1,\ldots,L_N\}} Error\left(d_{HL_i}, \hat{d}_{HL_i}\right). \tag{2.6}$$

where $Error(\cdot)$ is the same as above (Eqn. 2.5).

Fig. 2.4(b) illustrates the positioning of an ordinary hosts on a 2-dimensional space. The coordinates depends on the measured distances to the landmarks and their coordinates.

Ng and Zhang have remarked that GNP may be extremely inaccurate with respect to the directional relative error but proposes high rank accuracy. Thus, GNP is a good model for nearest server selection.

2.1.3 Lighthouses

Pias et al. observe that the basis of the space of GNP must be formed by well-known pivot nodes: The landmarks [1]. In addition, every *joining node* must contact the same nodes that implies a lack of scalability and performances issues. *Lighthouses* propose to fix these issues.

Lighthouses is built above the concept of multiple *local basis* with a *transition matrix* $\mathbf{P}$. A basis of a geometric space $\mathcal{S}$ is a subset of linearly independent vector that generates $\mathcal{S}$ [21]. In GNP, every node computes its coordinates in the same basis (defined by the position of the landmarks). In Lighthouses, every nodes has its own basis – the local basis – and computes its coordinates in this basis. To compare the coordinates of two nodes, the position must be expressed accordingly to the same basis. The transition matrix permits to express the coordinates of a node A in the local basis of another node B. The issue of GNP is then overcomes since any host is able to determine its coordinates relatively to any set of pivot nodes. In other words, the nodes in Lighthouses do not need to use the same landmarks.

When a new host H_i joins the system, it finds an *entry point* H_j, i.e., any node that is already in the system. Node H_j provides H_i a list of nodes that can act as H_i lighthouse. The joining node selects $d+1$ arbitrarily nodes among those in this list and then constructs a local basis $\mathbf{L} = \vec{l_1}, \ldots, \vec{l}_{d+1}$ where each vector $\vec{l_i}$ refers to a pair of lighthouses. This local basis spans the virtual space. The set of lighthouses may differ from host to host. Lighthouses are the equivalent of landmarks in other systems.

If there is less than $d+1$ nodes when H_i enters the system, H_i constructs its local basis with all the nodes that already joined. The first basis is built after $d+1$ nodes have joined the system. The assumption is made that d is small compared to the number of nodes the whole system.

Coordinates computation

Every node must compute its own coordinates relative to its local basis. So that, a joining node must first compute the coordinates of the local basis it will use. The node first constructs an arbitrary vector set which is a basis for $\mathcal{S}$. It then applies the *Gram-Schmidt* process on this basis to construct its local basis. The purpose of Gram-Schmidt process is to convert a set of vectors in a set of orthogonal vectors which are generators of the same space [21].

The basis $\mathbf{v} = \vec{v}_1, \ldots, \vec{v}_{d+1}$ is transformed into the orthogonal basis $\mathbf{e} = \vec{e}_1, \ldots, \vec{e}_{d+1}$ using Gram-Schmidt as follows:

$$\begin{aligned} \vec{u}_1 &= \vec{v}_1, & \vec{e}_1 &= \tfrac{\vec{u}_1}{\|\vec{u}_1\|} \\ \vec{u}_2 &= \vec{v}_2 - \vec{v}_2 \perp \vec{u}_1, & \vec{e}_2 &= \tfrac{\vec{u}_2}{\|\vec{u}_2\|} \\ \vec{u}_3 &= \vec{v}_3 - \vec{v}_3 \perp \vec{u}_1 - \vec{v}_3 \perp \vec{u}_3, & \vec{e}_3 &= \tfrac{\vec{u}_3}{\|\vec{u}_3\|} \\ & \ldots & & \\ \vec{u}_{d+1} &= \vec{v}_{d+1} - \sum_{i=1}^{d} \vec{v_{d+1}} \perp \vec{u}_i, & \vec{e}_{d+1} &= \tfrac{\vec{u}_{d+1}}{\|\vec{u}_{d+1}\|}. \end{aligned} \tag{2.7}$$

where $\vec{v} \perp \vec{u}$ is the projection of vector $\vec{v}$ orthogonally onto the vector $\vec{u}$.

To estimate distances, coordinates of any two hosts must be expressed in the same basis. The conversion between basis is provided by the application of the transition matrix $\mathbf{P}$. Lighthouses allows nodes to choose their local basis arbitrarily provided they preserve the invariant of maintaining a transition matrix $\mathbf{P}$. This matrix permits the transition from the local basis to a *global basis* $\mathbf{G}$. Lighthouses proposes a technique to construct a consistent transition matrix. The particularity of this approach is that it does not require to refer to a well-known global basis.

In linear algebra, the coordinates of a vector $\vec{x}_{\mathbf{B}}$ expressed in a basis $\mathbf{B}$ of a vector space $\mathcal{S}$ of size d become $\vec{x}_{\mathbf{B}'}$ in an other basis $\mathbf{B}'$ of the same space using the following conversion:

$$\vec{x}_{\mathbf{B}'} = \mathbf{P}^{-1}\vec{x}_{\mathbf{B}}. \tag{2.8}$$

where the columns of $\mathbf{P}$ are the coordinates of the new basis relative to the old basis. The column vector of $\mathbf{P}$ are:

$$\mathbf{P} = \begin{bmatrix} \vec{u}'_{1_{\mathbf{B}}} \\ \cdots \\ \vec{u}'_{d_{\mathbf{B}}} \end{bmatrix}.$$

where $\vec{u}'_{i_{\mathbf{B}}}$ is the i-th component of the basis $\mathbf{B}'$ relative to the basis $\mathbf{B}$.

To compute its transition matrix $\mathbf{P}$ between its local basis and the global basis, H_i does only require the transition matrix of its entry point.

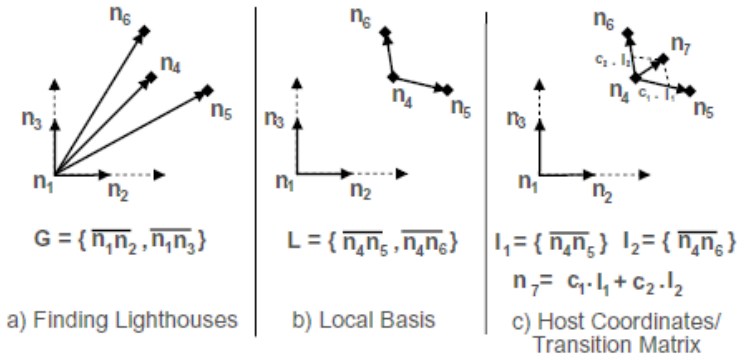


Figure 2.5: The three computation steps in Lighthouses (presented in [1])

When a node H_i with a transition matrix $\mathbf{P}_i$ wants to work on the coordinates $\vec{c}_{jl_j}$ of a node H_j with a transition matrix $\mathbf{P}_j$, it has to convert coordinates $\vec{c}_{jl_j}$ of H_j expressed in the local basis of H_j to coordinates $\vec{c}_{jl_i}$ of H_j expressed in the local basis of H_i using:

$$\vec{c}_{jl_i} = \mathbf{P}_i \mathbf{P}_j^{-1} \vec{c}_{jl_j}. \tag{2.9}$$

Fig. 2.1.3 illustrates the different steps Lighthouses follows to determine the coordinates and transition matrix of a node. The calculi of local basis, nodes coordinates and transitions matrices are presented in detail in [1].

2.1.4 Big-Bang Simulation (BBS)

Shavitt and Tankel discover that the accuracy of IDMaps was dependant of the positions between hosts and Tracers [9]. Worst, IDMaps is only able to find the closest node in 85% of the cases, that is far from being acceptable [9].

Based on the idea of coordinate systems proposed by Ng and Zhang, Big-Bang Simulation (*BBS*) models the network nodes as a set of particles. Each particle is the image of a node in a Euclidean space. Particles are traveling in the space under the effect of *potential force field*. The name "Big-Bang Simulation" comes from the fact that particles are initially placed at the origin.

The field force $\vec{F}_{i_0}$ (Eqn. 2.11) is derived from potential energy E_T which is the total error presented in Eqn. 2.10.

$$E_T(H_1, \ldots, H_N) = \sum_{i,j=1|i>j}^{N} Error\left(d_{H_iH_j}, \hat{d}_{H_iH_j}\right). \tag{2.10}$$

$$\vec{F}_{i_0} = - \bigtriangledown_{\vec{v}_{i_0}} E_T(\vec{c}_{H_1}, \ldots, \vec{c}_{H_N}). \tag{2.11}$$

Where $\bigtriangledown_{\vec{x}} f(\cdot)$ is the gradient of the function f with respect to vector $\vec{x}$.

The potential energy is the same as the objective function of landmarks in GNP (Eqn. 2.4) where the sum is on all nodes instead of being on all landmarks.

The force field reduces the potential energy of particles and particles pull or repulse others depending on the error of the distance between them [22].

The position and velocity of particles at time $t + \delta t$ are calculated by applying Newton's law of motion and the new potential energy is calculated at the end of the iteration. Increasing the *timestep* δ provides greater numerical efficiency. On the contrary, a small timestep permits to attract particles to a global minimum potential energy. A good introduction to Newton's law of motions can be found in [22].

If the particles were only under the affect of the force field, they would move away too fast and the particles would oscillate forever with constant velocity. To solve this problem, a *friction force* is added. With this force, a part of the energy is lost due to friction and the system can stabilize. The friction force depends on the normal force of the particle. The moving particles are assigned a friction coefficient μ_k and the static particles are under the effect of the μ_s friction coefficient.

The magnitude F_{ij} of the field force determines how much the induced force pulls or repulses the two particles i and j. A positive value means that the force pulls the two particles together. On the contrary, for a negative value, the two particles are repulsed. Shavitt and Tankel have shown that this induced force is given by the derivative of the *prediction error*. The prediction error is the embedding error between two nodes (particles in BBS) with respect to the Euclidean distance between the two nodes.

The previous NCS we have seen use conventional gradient minimization schemes, i.e., the *down hill simplex* (*DHS*) algorithm [23]. When DHS is used, the minimization can be caught by a local minimum but a local minimum is not necessarily the global one. Thus, while traditional coordinate systems running DHS are very sensitive to the initial coordinates, BBS does not care about initial coordinates. This quality of BBS is the result of the kinetic energy accumulated by the moving particles that permits them to escape a local minimum.

BBS achieves equally good performances for a wide range of system friction coefficient and is insensitive to small changes in the input graphs, e.g., the resuming of the simulation after a 20% variation of inputs needs less than 5% of the simulation time starting at the origin [9].

2.1.5 Practical Internet Coordinates (PIC)

Some coordinate systems rely on a small set of nodes – the landmarks – to determine the coordinates of all the nodes. Others use sets of nodes picked at random as landmarks. This solution is scalable but is difficult to secure. On the contrary, the first solution is easy to secure but not scalable.

PIC (for Practical Internet Coordinates) proposes a solution: It does not rely on infrastructure nodes but can computes accurate coordinates even when some peers are malicious. The protection is based on a test of the triangular inequality which is the base of the system. Even if it has been demonstrated [24, 25, 4] that the triangular inequality is commonly violated

on the Internet, PIC is a good introduction to security problems in coordinate systems. We present the security methodology of PIC in Chapter 3.

Like others coordinate systems, PIC maps each node to a point in a d-dimensional Euclidean space. A joining node picks at least $d+1$ nodes already in the system as landmarks and then computes its coordinates based on landmarks ones. The set of landmarks is written L and can be chosen using the three following strategies:

Random: The elements of L are randomly picked.

Closest: The elements of L are the elements closest to the node in the network topology (i.e., the smallest distances).

Hybrid: Some elements of L are randomly picked and others are the closest.

Experiments in [10] show that the hybrid strategy offers the lowest relatives errors. However, the random strategy works better for long distances and the closest strategy is better for short distances.

Let's go back to the closest strategy to choose L. Traditionally, to find the closest node, H_n picks a node H_c at random among other nodes. Then, it probes the distances to all of H_c neighbours and picks the closest neighbour. If a neighbour of H_c is closer than H_c, H_c is updated to point to this node and the process is repeated. Otherwise, the algorithm stops and H_c is the "closest" node to H_n. If the k closest nodes must be kept, the algorithm can keep track of these k closest nodes. However, it is possible to reduce the overhead of the method by using estimated distances rather than probes. Meanwhile, a problem occurs when the node has no coordinates yet. The solution is to use the random strategy on joining nodes to generate an estimate of the coordinates and then use the coordinates to find the closest nodes. When the closest nodes are found, the coordinates are refined using the hybrid strategy. The reader is invited to read [10] for more details about the algorithm.

A global optimisation algorithm computes the new coordinates for all the nodes. As presented in others methods this optimisation minimizes the predicated distance error. The DHS algorithm optimizes the following relative error estimate (the target function of DHS):

$$\sum_{j=1}^{|L|} \left(\frac{d_{H_n H_j} - \hat{d}^{\mathcal{S}}_{H_n H_j}}{d_{H_n H_j}} \right)^2 . \tag{2.12}$$

If there is less than $d+1$ nodes, H_n selects all the nodes and obtains the all-pairs distances. The global optimization algorithm is then applied on the set.

2.1.6 Vivaldi

We have seen before that the global error of coordinate systems can be computed as follow:

$$E = \sum_{i,j|i\neq j} Error(d_{H_i H_j}, \hat{d}_{H_i H_j}). \tag{2.13}$$

Dabek et al. noticed that an analogy could be done between this global error and physical mass-spring systems potential energy. Thus minimizing E is the same as minimizing the potential energy of a virtual network of springs. Based on this observation, Dabek et al. proposed *Vivaldi* [2].

Vivaldi is a fully decentralized network coordinate system which makes no distinctions between nodes. A Vivaldi node collects distance information for a couple of neighbours and computes its new coordinates with the collected measurements. The idea is that every node i is represented as an unitary mass connected to each neighbour j by a spring with the rest length set to the measured RTT (d_{ij}). The actual length of the spring is the distance ($\hat{d}_{ij}$) predicted by the coordinates space. A spring always tries to have an actual length equals to its rest length. Thus if $\hat{d}_{ij}$ is smaller than the measured RTT, the spring pushes the two masses attached to it. On the opposite, if the spring is too long, it pulls the masses and reduce its actual length. The coordinates in Vivaldi are updated following this principle. If we note $\vec{c}_i$ the coordinates of i and $\vec{c}_j$ the coordinates of j, the new coordinates is computed as follows:

$$\vec{c}_i = \vec{c}_i + \delta \cdot \left(d_{ij} - \hat{d}_{ij} \right) \cdot u\left(\vec{c}_i - \vec{c}_j\right). \tag{2.14}$$

which must be understood as the displacement of the mass by a small part of the displacement induced by the spring applying the Hook's law.

Remember that the Hook's law gives the force $\vec{F}_{ij}$ that the spring between mass' i and j exerts on mass i [22]:

$$\vec{F}_{ij} = \left(L_{ij} - \|\vec{x}_i - \vec{x}_j\|\right) \cdot u\left(\vec{x}_i - \vec{x}_j\right). \tag{2.15}$$

where L_{ij} is the rest length of the spring, $\vec{x}_i$ is the coordinates of mass i and $\vec{x}_j$ the coordinates of mass j.

This force defines if the spring pulls or pushes i and j. Fig. 2.6 illustrates the effect of the Hook's low on a system with two masses i and j connected by a spring. In Fig. 2.6(a) the actual length of the spring is too small compared to the rest length. Thus, by the Hook's law, the spring pushes the two masses. When the system is stabilized, the actual length equals the rest length as depicted in Fig. 2.6(b).

δ, the adaptative timestep, defines the fraction of the way the node is allowed to move towards the perfect position for the current iteration. The timestep depends on the local errors of the two nodes (e_i and e_j) and permits to limit the displacement if the error is important. The timestep is defined as follow:

$$\delta = c_c \cdot \omega. \tag{2.16}$$

where c_c is a tuning constant and $\omega = e_i/(e_i + e_j)$.

When a node has computed its new coordinates, it computes its local error with Eqn. 2.17.

(a) Before the application of the Hook's force

(b) After the effect of the Hook's force

Figure 2.6: Effect of the Hook's force on a two masses-spring system

$$e_i = e_s \cdot \omega + e_i \cdot (1 - \omega) \tag{2.17}$$

where e_s is the *relative error* defined in Eqn. 2.18. $u\left(\vec{x}_i - \vec{x}_j\right)$ gives the direction of the displacement of i and is normalized to 1.

$$e_s = |d_{ij} - \hat{d}_{ij}|/d_{ij}. \tag{2.18}$$

The relative error used in Vivaldi is not exactly the same as the relative error proposed by Ng et Zhang and presented in Eqn 2.2.

Eqn. 2.14 is the core of Vivaldi since it allows nodes to discover their coordinates based on the observations of other nodes.

Dabek et al. propose to use *height vectors* instead of Euclidean space for the virtual space. A height vector consists of an Euclidean coordinates augmented by a height. The Euclidean part of the vector models the Internet backbone where latencies are proportional to geographic distances. The height models the time the packets take to travel the access links to the backbone. The height depends on the queuing delay, the link quality, etc.

The height space redefines the following operations:

$$\begin{aligned} [x, x_h] - [y, y_h] &= [(x - y), x_h + y_h] && (2.19)\\ \|[x, x_h]\| &= \|x\| + x_h && (2.20)\\ \alpha \cdot [x, x_h] &= [\alpha \cdot x, \alpha \cdot x_h]. && (2.21) \end{aligned}$$

where Eqn. 2.19 computes the vectorial difference, Eqn. 2.20 computes the norm of a vector and Eqn. 2.21 computes a scalar product with a vector.

The fundamental difference between adding a height to a Euclidean space and adding a dimension is that even if two vectors have the same height, the distance between them is the

euclidean distance plus the two heights. We have also to notice that the height is always additive, even in subtraction (Eqn. 2.19).

When a node is not able to move in any direction, the force pushes the node up away from the Euclidean space and the height increases.

Authors of Vivaldi have shown that the better results are obtained with the height vector model. Nevertheless, Euclidean spaces or spherical coordinate spaces also offer accurate coordinates.

Vivaldi converges to stable accurate coordinates when the triangular inequality is respected. Unfortunately, it is far common on the Internet to violate the triangular inequality [24, 25, 4] and de Launois demonstrated that sometimes coordinates in Vivaldi do not converge even for systems with only three nodes [11]. This is why he proposes an adaptation of Vivaldi: *SVivaldi*. SVivaldi offers two modifications to Vivaldi: (*i*) a better computation of local error estimate and (*ii*) a loss factor is added to simulate the loss of energy due to friction in the springs and thus avoid oscillations.

In Vivaldi algorithm, at each step, the local error is modified to match the error for the node being sampled. Consequently, much importance is given to the last measured RTT and the local error depends on the neighbour considered. Thus, the local error estimate may vary from time to time depending on the neighbour. SVivaldi proposes a more accurate local error estimate. The node retains the last prediction error computed for each of its neighbours, and computes the local error estimate as the average of those prediction errors.

The new local error estimate proposed by SVivaldi improves the accuracy of the nodes but does not prevent the system from oscillating. This is why SVivaldi also introduces a *loss factor*. The idea is that, at the beginning of the algorithm, nodes can move easily and this ability decreases with the time. The analogy can be made with real springs where some energy is lost due to friction preventing infinite oscillations. This loss factor $loss_i$ is introduced by modifying the timestep presented in Eqn. 2.16. The SVivaldi's timestep is computed as follow:

$$\delta = c_c \cdot \omega \cdot (1 - loss_i)\,. \tag{2.22}$$

where $loss_i \in [0, 1[$ is the loss factor to neighbour i and is computed by Eqn. 2.23.

$$loss_i = c_f + (1 - c_f) \cdot loss_i. \tag{2.23}$$

where c_f is a tuning parameter.

Fig. 2.1.6 presents the evolution of the loss factor for a particular node to its neighbour a. With this curve and Eqn. 2.22, we can observe that the part of the Vivaldi timestep used for the calculus of coordinates decreases with the time in SVivaldi. If an important change is observed in the topology the loss factor must be reinitialized to a lower value.

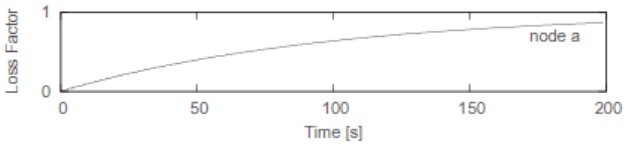


Figure 2.7: Evolution of the loss factor with time to a node a (presented in [2])

2.1.7 Network Positioning System (NPS)

Ng and Zhang identify in [3] the three key issues for coordinate system. First, the network positions must be consistent. Second, the positions must adapt to topology changes. Finally, the positions must remain stable when there is no change in the topology. They propose NPS, a *hierarchical* decentralized network coordinate system that maintains coordinates consistency and accuracy.

NPS computes coordinates like GNP with the objective functions Eqn. 2.4 and Eqn. 2.6 to optimize. However, in GNP there is two kinds of active nodes, the landmarks and the ordinary hosts. NPS adds a new type of nodes: The *membership servers*. A description of the three sort of nodes in NPS is presented below:

The membership servers that store systems information and maintains soft state about some participants.

The landmarks that are used to define the basis of the Euclidean space and that can be used as reference points.

The ordinary hosts that are any other node in the network.

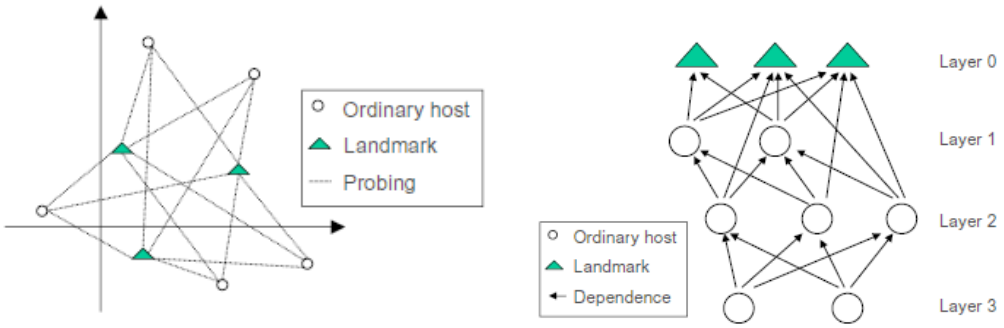
In NPS, every ordinary node might choose its reference points. So that the landmarks are not always the references for nodes. Nevertheless, the landmarks compose the basis for the space and coordinates of nodes are always expressed in this basis.

Even if NPS is based on landmarks, it remains scalable and robust to temporary landmarks failures. Indeed, NPS is built on a hierarchical architecture. Landmarks define the basis and can serve as reference points. But every node that has already determined its position can also act as a reference point. To compute its coordinates, a node needs at least $d + 1$ references. The list of random references for a node is given by the contacted membership server among a list of *eligible nodes*. A node is eligible if it is in the system and has coordinates.

Before continuing the presentation of NPS, we must introduce the concepts of *dependency* and *layer number*:

Dependency: There exists dependency between A and B if A uses B as one of its reference points.

Layer number: The layer number of a host is the maximum number of dependency hops separating it from the landmarks, landmarks always have level 0.



(a) Basic landmarks method (b) Hierarchical method

Figure 2.8: Coordinate systems architectures (presented in [3])

An eligible host must have already determined its position and must have a lower layer number that the node using it. When a joining node starts, it sets its layer number to the highest layer number allowed by the system, L_{max} given by the membership server. It then receives a reference points set L such that all references have always lower layer number. Based on this set, the node updates its layer number L_H to the maximum of the layer number of hosts in the set plus one. After, (i) the node probes references and (ii) computes its own position. Next, (iii) it repeats steps (i) and (ii) until its coordinates are stabilized.

Fig. 2.8(b) presents the architecture and dependencies between nodes in NPS. With this illustration, we can see that the load on landmarks is not a function of the participants in the system. At the opposite, Fig. 2.8(a) presents the dependencies in a traditional landmarks-based coordinate system. In the last, the load of landmarks is proportional to the number of participants in the system.

Ng and Zhang have shown that a 3-layer is sufficient for most of the applications [3]. Indeed, with an acceptable bandwidth of 1Mbps dedicated for landmarks to support layer 1 nodes and 10kbps for layer 1 nodes to support layer 2 nodes, the system can support up to 2 millions nodes at layer 1 and up to 2 billion nodes at layer 2.

To maintain stability, landmarks cooperate and move only if the position change by more than one percent comparing to the old position.

2.2 Conclusion

In this Chapter we proposed a brief overview of the most well-known latency estimators and saw that coordinate systems are an interesting solution for estimating latencies between any pair of nodes without having to probe them first.

Table 2.2 presents a summary of the most important functionality of latency estimators presented before. In this table, '○' means the system does not support the functionality, '●' means the functionality is supported, '◔' means a partial support and '–' is noted when not

	IDMaps	**GNP**	**Lighthouses**	**BBS**	**PIC**	**Vivaldi**	**NPS**
Coordinate system	○	●	●	●	●	●	●
Infrastructure	●	●	○	○	○	○	●
Landmarks	–	●	○	○	○	○	·
Scalable	·	○	●	●	●	●	●
Iterative	○	○	○	●	○	●	○
Embedded security	○	○	○	○	●	○	●
Accurate	○	○	·	●	●	●	●

Table 2.1: Comparison of latency estimators systems

applicable. Regarding the accuracy, a '○' means that the system is less accurate than a system with '·'. In the same manner, a system with a '●' is more accurate than a system with '·'.

We evaluate the systems presented above with 7 points. The first and obvious criterion is to define if the system uses a coordinate space. After, we see if the system rely an infrastructure. Another important property of the system is the presence of landmarks. A good distance estimator system must also be scalable and support thousands of nodes. Next, we highlight systems which compute coordinates iteratively. An iterative system refines the coordinates gradually. Finally, we look for systems with embedded security and classify them by the accuracy of the coordinates. This last point is a bit subjective because no work has been done to compare the coordinate systems we have presented above.

With the summary, we can observe that BBS, Vivaldi and NPS are the most interesting coordinate systems. However, the BBS technique is a bit difficult and looks theoretic. Vivaldi is fully decentralized and needs no infrastructure. So that, Vivaldi does not have single point of failure. Unfortunately, it is quite hard to secure Vivaldi due to the equity of nodes. NPS is based on an infrastructure of landmarks and membership servers. This solution permits to be robust to attacks on coordinates but if attackers decide to target the infrastructure, NPS may become inaccurate. In Chapter 3 we see the different security issues in coordinate systems and propose a solution to fix them in Chapter 4.

For the rest of the work we refer to Vivaldi and NPS for securing coordinate systems. However, we mainly focus on Vivaldi because techniques for securing a decentralized system also permit to secure an infrastructure-based system like NPS while the opposite is not always true.

3 Attacks on Network Coordinate Systems

3.1 Introduction

It has been demonstrated that network coordinate systems are sensible to many different attacks due to their characteristics [4, 16]. Indeed, coordinate systems are always-on globally distributed large-scale services. So that if an intruder arrives to break the system, its misdeed may concern a wide range of users.

Position of a single node in a NCS is based on the coordinates of other nodes which are trusted and considered to be accurate. Unfortunately, coordinates of a neighbour may be fake or a reported delay may be completely wrong.

In this section we first introduce security concerns of coordinate systems with the presentation of the embedded security in PIC and NPS. After this presentation, we determine the different attacks coordinate system are sensible to (based on [4, 16]). Afterwards, we analyze the different attacks and illustrate how attackers may lead these attacks.

3.2 Earlier security protection in coordinate systems

In Chapter 2, we have seen several coordinate systems. Two of them, PIC and NPS, propose a solution to security concerns. In this section, we present these security issues and the solutions PIC and NPS address to them.

3.2.1 Security in PIC

PIC was the first coordinate system to propose an embedded security protection. Costa et al. remarked that if a malicious node is selected as a landmark, it can lie about its coordinates or/and interfere with distance measurements [10]. The result of this attack is that a set of coordinates can be arbitrarily wrong. The triangular inequality would be violated after such an attack. To solve this security problem, the system can verify the triangular inequality. When a node detects the triangle inequality is violated, it checks the inequality for every of its

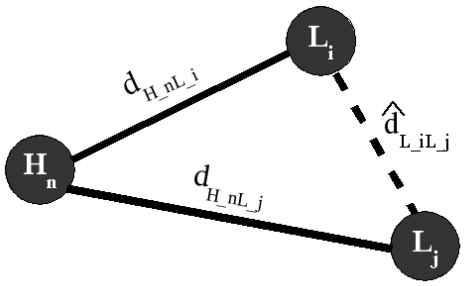


Figure 3.1: Triangular inequality illustration

landmarks and remove the node that most violate the triangular inequality from L, the set of landmarks.

For most of the triple of nodes A,B and C, Costa et Al. suppose the following triangular inequality property matches [10]:

$$\begin{cases} d_{AB} + d_{BC} \geq d_{AC} \\ \hat{d}_{AB} + \hat{d}_{BC} \geq \hat{d}_{AC}. \end{cases}$$

For the general topology presented in Fig. 3.1 where H_n is a node and L_i, L_j two distinct landmarks of H_n. The following properties should hold due to triangular inequality:

$$d_{H_nL_i} \leq d_{H_nL_j} + \hat{d}_{L_iL_j} \tag{3.1}$$
$$d_{H_nL_i} \geq d_{H_nL_j} - \hat{d}_{L_iL_j} \tag{3.2}$$
$$d_{H_nL_i} \geq \hat{d}_{L_iL_j} - d_{H_nL_j}. \tag{3.3}$$

Inequality Eqn. 3.1 imposes an upper bound on the measured distance $d_{H_nL_i}$ and inequalities Eqn. 3.2 and Eqn. 3.3 impose a lower bound of $|d_{H_nL_j} - \hat{d}_{L_iL_j}|$.

For each landmark L_i in L, the security test checks that the bounds are respected with the following two metrics:

$$upper_{L_i} = \sum_{L_j \in L} \begin{cases} d_{H_nL_i} - \left(d_{H_nL_j} + \hat{d}_{L_iL_j}\right) & \text{if } \left(d_{H_nL_j} + \hat{d}_{L_iL_j}\right) < d_{H_nL_i}, \\ 0 & \text{otherwise.} \end{cases} \tag{3.4}$$

$$lower_{L_i} = \sum_{L_j \in L} \begin{cases} \left|d_{H_nL_j} - \hat{d}_{L_iL_j}\right| - d_{H_nL_i} & \text{if } \left|d_{H_nL_j} - \hat{d}_{L_iL_j}\right| > d_{H_nL_i}, \\ 0 & \text{otherwise.} \end{cases} \tag{3.5}$$

The $upper_{L_i}$ metric is the total deviation above the upper bound for landmark L_i and the $lower_{L_i}$ metric is the total deviation below the lower bounds for the same landmark.

The maximum value of the two metrics are computed and the landmarks that reach that value are removed from L. The node use an optimisation algorithm on target function Eqn. 2.12 with the remaining landmarks. This process can be repeated until the predicated distance between the node and its landmarks is below a fixed threshold.

PIC proposes a solution for security issues based on the triangular inequality. Even if this solution is not acceptable due to frequent violations of the triangular inequality on the Internet [25, 24]. Authors of PIC have the merit to be the first to propose a secure approach for coordinates system.

Security in NPS

Authors of NPS discovered that malicious nodes can lie about their coordinates with two types of lies: (1) *continuously changing lies* and (2) *fixed lies*.

The damage of continuously changing lies is limited. Indeed, if the reference point continuously changes its positions, its dependent can detect it. The solution is to put a time limit within which the reference point must stabilize. The distance to the reference node must also stabilize within a fixed time limit.

Fixed lies are more dangerous because it is hard to detect. Fixed lies malicious nodes consistently reports the same false position and/or network distance. The proposed approach is to eliminate reference points which fit poorly in the Euclidean space compared to other reference points. The *fitting error* of a reference $R_i \in L$ of an ordinary node H_n is computed as follow:

$$E_{R_i} = \frac{|\hat{d}_{H_n R_i} - d_{H_n R_i}|}{d_{H_n R_i}}. \tag{3.6}$$

The estimated distance $\hat{d}_{H_n R_i}$ to the reference R_i is evaluated with the coordinates based on the $d+1$ reference points. If the criterion presented in Eqn.3.7–3.9 are reached for the reference point with fitting error E_{R_i}, the reference point is no more considered as a valid reference point.

$$E_{R_i} = \max_{R_j \in L} \left(E_{R_j}\right) \tag{3.7}$$

$$E_{R_i} > f_e \tag{3.8}$$

$$E_{R_i} > C \cdot median_{R_j \in L} \left(E_{R_j}\right). \tag{3.9}$$

f_e and C are tuning constants in Eqn. 3.7–3.9. Eqn. 3.7 means that R_i has the worst fitting error among all the reference points. Secondly, Eqn. 3.8 ensure that the fitting error of R_i is greater than a given threshold. Finally, Eqn. 3.9 impose that E_{R_i} must be greater than a multiple of the median of all the fitting errors of all reference points.

In other words, a reference point is rejected if its error is large enough and if it is significantly larger than the median of the errors of all the reference points.

The solution proposed by NPS requires to set the thresholds a priory. Moreover, this technique can give false-positive (i.e., a reference is considered as bad even though it is good) or false-negative (i.e., a malicious node can be considered as good using this test).

PIC and NPS both propose a solution to protect the system from misbehaving nodes. Unfortunately, none of these solutions is effective for the Internet. In the next section, we present the real attacks coordinate systems are sensible to. Afterwards, in Chapter 4, we propose a formal model to secure coordinate system in general and validate it in Chapter 5.

3.3 Attacks

Coordinate systems often support thousands of nodes. Nodes can be so numerous that it is impossible to secure all of them. As a result, the attack can come from the inside. Typically, coordinate systems cannot rely on nodes acting in the system and must consider that anyone can be dangerous. This property is really important because it cancels traditional protection like simple cryptography. Cryptography cannot protect against a fake content given by a trusted participant.

Another problem with large-scale systems is that attacks can be the result of a group of attackers and not only the result of an isolated node.

In this text, we classify attacks on coordinate systems in four different types:

Denial of Service (DoS): Malicious nodes try to alter performances of the system.

Take the control: Malicious nodes try to take the control of the system (become a landmark).

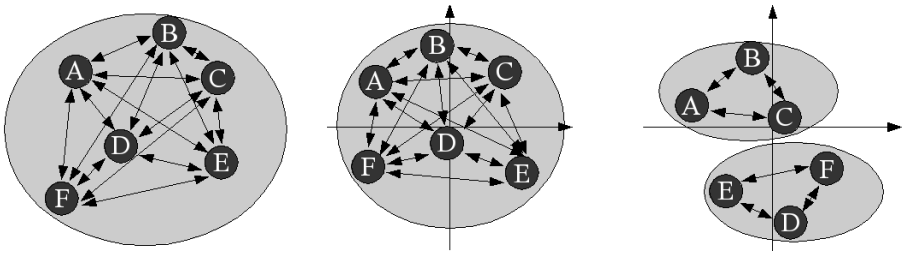
Man-in-the-middle attack: Malicious nodes may eavesdrop, drop or change the content of messages.

NCS as a backdoor: Like any other service running on a host, NCS may become a backdoor for worms propagation or may carry an other attack.

Except for the backdoor category, the result of the attack is a deformation of the coordinates space. Many techniques can be used to change the behaviour of a coordinate system. The attacks can modify the results of measurements, alter the global bandwidth for particular paths or even alter coordinates of some nodes. Most of the time, the last attack causes the neighbours of the attacked nodes to move and also the neighbours of neighbours and so on.

When the coordinates space is deformed, the predicted distances may become wrong and the different choices of services based on the NCS may become inconsistent.

In the next two sections, we first analyze the attacks coming from the outside. After, we see how an internal attack can be conducted.



(a) Real network (b) Virtual network before attack (c) Virtual network after attack

Figure 3.2: External attack: the drop of messages can split a single space into two independents spaces

3.3.1 External attacks

The *external attacks* are those that can be conducted by nodes outside the system. A node which wants to do such an attack does not need to have coordinates or to be an actor in the choice of coordinates. However, the external attacks can also be performed by nodes inside the coordinate system.

In this section, we consider three external attacks: (*i*) alteration of the content of messages, (*ii*) injection of fake messages and (*iii*) alteration of RTT measures.

Alteration of the content of the messages

External attackers may alter the content of NCS messages. One way to alter a message is to modify its content. Another solution is to drop the message.

To modify the content of a message, the attacker must see the message. Such attack looks not realistic because the attacker must be able to force messages to pass through it or an accomplice node. The drop of messages looks more realistic. Indeed, a malicious node does not require to see the message to avoid it to be delivered. The malicious node can attack the underlying topology with a DoS or any attack that can cause drop of messages in routers along the path of the message. However, this last attack requires to know the path followed by the target messages.

If all the messages for and from a particular nodes are dropped, this node appears unavailable in the system. As a result, a good distributed attack of this kind may divide a single NCS in two or more independent NCS.

Fig. 3.2 presents the alteration of communications with a drop messages attack. In this attack a set of outside nodes drop every message from any node in $\{A, B, C\}$ to any node in $\{D, E, F\}$ and every message from any node in $\{D, E, F\}$ to $\{A, B, C\}$. The result is that two distinct virtual spaces $\{A, B, C\}$ and $\{D, E, F\}$ are built instead of a single $\{A, B, C, D, E, F\}$ one. With this example, we show that, only by dropping NCS messages, it is possible to isolate a part of a virtual topology and thus disallowing communications between some nodes that are still connected. In addition to the division of the virtual topology, a drop message attack can be used to impose traffic to pass through a specific node.

Instead of dropping messages, a malicious outside node may change the content of messages and, more precisely, the coordinates specified originally in the message. Then even if every node in the virtual topology is safe and gives good information, the resulting coordinates may be wrong. The result of the alteration attack is a DoS. Indeed, the coordinates may be incorrect and thus the distance estimation may be wrong and an incorrect node may be chosen, resulting in a lack of performances.

The alteration of message can be classified in the man-in-the-middle attack category.

Injection of fake messages

It is possible for an external node to inject fake messages inside a coordinate system. Injection of messages can be classified in two categories.

An external node can send messages to nodes in the NCS without any intelligence. An outside node may also eavesdrops query messages in the virtual network and sends fake answers to requests.

The first attack injects query messages to overload the system and thus slow down performances of services based on the system. The second attack is more intelligent and the result is the same as the "alteration of the content" attack. In both case, the result is a DoS.

If messages in the coordinate system are too predictable, the malicious node can predict when a particular message is sent and thus gives a fake answer before the legitimate answer. If the system is really too predictable, malicious nodes do not even need to eavesdrop the messages.

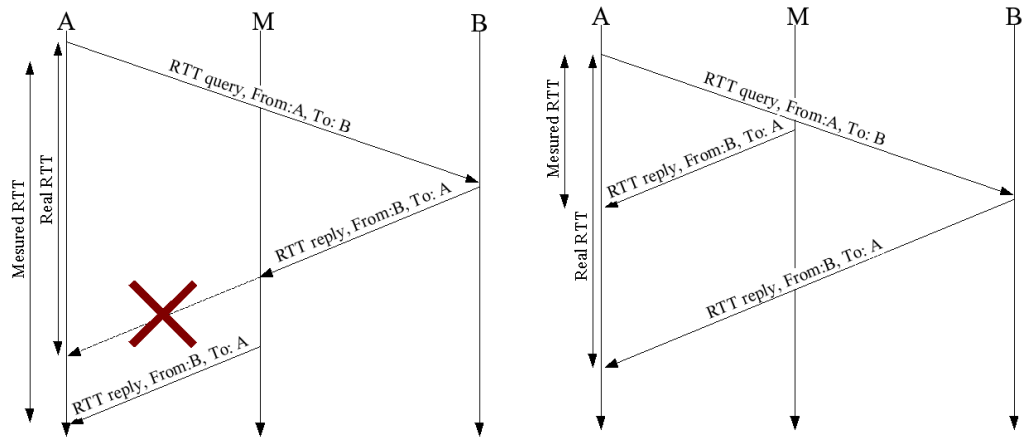
If the coordinate systems are not correctly implemented, the injection of fake messages can be easy.

Fortunately, the injection of smart fake messages should be quite hard to implement in practice since it is not so easy to build a man-in-the-middle attack. However, Donnet and Bonaventure demonstrated that it was possible [17].

Alteration of RTT measurements

The accuracy of coordinates is dependent on the quality of round-trip measurements. Indeed, points in the coordinates are such that estimation error is minimized. And the estimation error is the difference between real distances and computed distances. Thus if an attacker can alter RTT, it is able to alter the accuracy of the whole system.

An exterior node can performs two different attacks to alter round-trip time measurements. (*i*) If an attacker is between the two nodes making the measurement it can increase the RTT by keeping RTT probe message for a long time. It is also possible for it to decrease the RTT by answering immediately instead of the real target. Nodes seem to be much far than they actually are when a RTT is increased artificially. When the measured RTT is lower than the real RTT, the node looks closest than reality. (*ii*) Another approach is to attack the underlying network and not the NCS directly. The attacker can periodically slow down a part of the network that



(a) Increased RTT with injection and drop of message (b) Decreased RTT with injection and drop of message

Figure 3.3: Modification of the RTT with the injection of a fake message

supports the NCS and thus RTT changes over the time and coordinates may have difficulties to converge. The simplest way to perform this attack is to send burst of packets to routers supporting the services and thus fill buffers and provoke loss of messages.

It is also possible to give bad RTT measurements while injecting fake answers to round-trip times probes. When a malicious node M detects that a good node A sends a RTT probe to B, the bad node can send A an answer prior to B. Then, A uses the bad reply to compute the RTT instead of waiting the answer from Donnet and Bonaventure have demonstrated that current implementations of coordinate systems are sensible to this attack [17].

Fig. 3.3 presents two possible procedures followed by a node M to alter the RTT between a node A and a node B. Fig. 3.3(a) presents how M increases the measured RTT. The bad node drops the reply (or the request) form the RTT and sends a reply much later. On the contrary, Fig. 3.3(b) shows how M decreases the measured RTT by replying the request for the RTT before B. A considers the measured RTT as the difference of time between the send of the query and the reception of the first reply to this query.

Attacks on RTTs measurements are not inevitably man-in-the-middle attacks. Even for the injection of fake answers, the malicious node does not require to see the traffic of its target node as we have seen in Sec. 3.3.1.

The alteration of round-trip times is a DoS attack.

In this section we have seen how nodes from the outside of a coordinate system can attack it. An external node can alter the measured distances between nodes, but it can also inject, alter or drop messages in the embedding system.

In the next section, we study the attacks that can be performed by nodes inside the system. Chapter 4 presents a solution to protect coordinate system against both external attacks we have seen and internal attacks presented in Sec. 3.3.2.

3.3.2 Internal attacks

While many of the external attacks can be thwart with simple solutions like *cryptography* and *nonces* [4], it looks hard to provide good protection against attacks performed by trusted nodes.

In this section we discover the different attacks that a member of a coordinate system can perform against its own system. A member of a NCS is called an *internal node* or *trusted node*. It is important to notice that an internal attack can be performed by an outside node that manages to be considered as internal, e.g., *spoofing* [26]. In addition, it is also possible for an internal attacker to perform the external attacks we have seen above.

Kaafar et al. proposed two performance indicators to estimate the impact of an attack on a coordinate system [4]. The relative error is the most important indicator. The more the relative error is, the worst the coordinates are. If the result of an attack gives higher relative errors than without the attack, the attack has a negative impact on the accuracy. The second indicator proposed is the *relative error ratio* (also called *ratio*). The ratio is the relative error measured during an attack normalized to the relative error in absence of attackers. A value under 1 for the ratio indicates an improvement of the accuracy. On the contrary, an error ratio above 1 indicates a loss of accuracy compared to the system without cheats.

Kaafar et al. propose the next four attacks and evaluate their impact on Vivaldi and NPS [4]. In the following, we summarize the attacks and the most important observations about them.

Disorder attack

The *disorder attack* has no specific objective, it only introduces chaos in the coordinates. When points are placed randomly in a coordinate system, the error is important and the result is a DoS. To increase the overall error, the attacker has just to maximize the relative error of nodes. To conduct such an attack, the malicious node can falsify its coordinates and delay distances probes. It may also not cooperate. The result of disorder attacks is high error in the predicted distances or the non-convergence of the algorithm.

Disorder attack on Vivaldi: In the fully-distributed algorithm Vivaldi, the nodes rely on cooperation of others in order to compute highly accurate coordinates. With such an architecture, it is easy to fool honest nodes. When a malicious node is solicited, it answers with random coordinates and a low local error. To improve the attack, the malicious node can also randomly delay the measurement.

In disorder attacks, the consistency of the given coordinates is not required. Indeed, the Vivaldi attacked node considers itself as a high error node since the attacker sends a low error. As a result, the adaptative timestep of the good node is high and its coordinates can be strongly changed by the attacker. Indeed, the good node moves to fit with the information given by the bad node.

Kaafar et al. evaluate the impact of this attack on the data set used by Dabek et al. when they introduced Vivaldi [2]. For the evaluation, they set the local error of misbehaving nodes to a low value of $e_j = 0.01$. The coordinates randomly proposed by the bad nodes are in the interval $[-50000, 50000]$ and a randomly generated delay in $[100, 000]$ ms is added to the RTT.

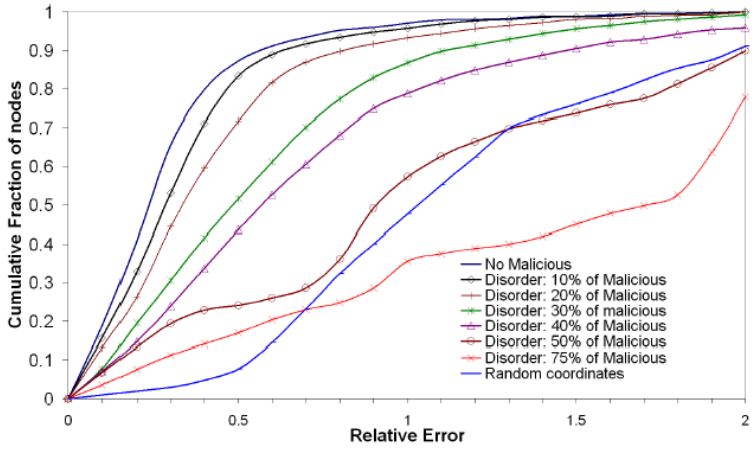


Figure 3.4: Disorder attack on Vivaldi: CDF of relative error after 5000 ticks [4]

Fig. 3.4 illustrates the evolution of the cumulative fraction of node with the relative error. We do not study this figure in detail in this section, but it is interesting to see that Vivaldi has globally higher relative errors in presence of attackers. When the number of attackers is too important, the error can even be worst than with arbitrarily chosen coordinates. We can conclude that Vivaldi is sensible to attacks since the relative errors of nodes are increased in presence of malicious nodes. We invite the reader to read [16] for a more detailed study of the disorder attack on Vivaldi or jump to Chapter 5 to see how we analyze the attack and solve it.

Isolation/Repulsion attack

In repulsion and isolation attacks, malicious nodes lie about their coordinates or the measured distances to move good nodes towards a predefined position or to reduce their own attractiveness.

In *isolation attack*, the malicious nodes try to convince the victims to move towards a zone they have decided. The purpose of such an attack is to force the victims to connect to an accomplice node as it looks to be the closest node. The accomplice node is then able to make traffic analysis, packets dropping or man-in-the-middle attacks.

In *repulsion attacks*, the malicious nodes try to alleviate their attractiveness or the attractiveness of a target. To this aim, the resource of the considered node are presented worst than they actually are. This attack can be performed using two different techniques, either by repulsing a set of targets far away from other nodes, or by repulsing all nodes away from a selected target. The result is the same: Some nodes are isolated in the virtual network.

To illustrate the last attack, we can imagine a service spread around the world where a particular node receives the same amount of money independently of the traffic passing trough it. Thus this node tries to isolate itself and then become less solicited (and thus reduce its bandwidth, processor or disk resources). On the contrary, if a node earns as much money as traffic passing through it (e.g., when the money is the result of advertisements), the node tries to be chosen by other nodes.

If this technique is used not as a cheat but with the approbation of all other nodes, it is called *traffic engineering* [27].

Technically, the isolation and repulsion attacks are the same. In the rest of this work, we equally use the term "repulsion attack" for both isolation and repulsions attacks.

It is possible to increase the impact of a repulsion attack by colluding the different attacks. In a *colluding isolation attack*, the malicious nodes attack a target (of one or many nodes) consistently in a collective way.

A first strategy could be to collectively move all other nodes from a chosen target. A second possible strategy could be to set the coordinates of the malicious nodes in a remote area and to try to capture a targeted node in the cluster.

Repulsion attack on Vivaldi: In repulsion attack, the malicious node chooses coordinates $\vec{c}_{\text{target}}$ where to move the target. $\vec{c}_{\text{target}}$ must be chosen far enough to allow *lie consistency*, i.e., the predicted and the measured distances must be equal after the attack. For this attack, malicious nodes know the current coordinates of their targets $\vec{c}_{\text{current}}$. Then malicious can compute the needed RTT's that are consistent with the lie:

$$RTT_{\text{needed}} = \left(\|\vec{c}_{\text{target}} - \vec{c}_{\text{current}}\|/\delta\right) + \|\vec{c}_{\text{target}} - \vec{c}_{\text{current}}\|. \tag{3.10}$$

where δ is the timestep computed by the target node (Eqn. 2.16).

Eqn. 3.10 computes the theoretical RTT that must be "measured" by the good node to move towards the target coordinates $\vec{x}_{\text{target}}$. If we consider that the malicious node knows the real RTT (RTT_{measured}), it can delay the measure by RTT_{delay} presented in Eqn. 3.11.

$$RTT_{\text{delay}} = RTT_{\text{needed}} - RTT_{\text{measured}}. \tag{3.11}$$

where RTT_{measured} is the measured RTT if no delay is added.

In this attack, each malicious node must randomly select their coordinates far away from the origin. The result is that target nodes are pushed far away from the origin at coordinates around $\vec{x}_{\text{target}}$ chosen by malicious nodes.

Fig. 3.5 illustrates the impact of repulsion attacks on the accuracy of coordinates in Vivaldi. Once again, we can observe that Vivaldi is sensible to attacks and the relative errors increases with the number of malicious nodes. Comparing Fig. 3.4 and Fig. 3.5 permits to observe that Vivaldi is more sensible to repulsion attacks than disorder attacks.

Fig. 3.6(a) depicts the evolution of the relatives errors of the target nodes for the two strategies of colluding isolation attack. This figure shows that it is more effective to move all the nodes away of a target (first strategy) than trying to capture a target into a remote area of the coordinates space (second strategy). This result can be explained as follow: Moving a large number of nodes away from their coordinates introduces much more error than moving only one node. This results is confirmed by Fig. 3.6(b) where the worst results of the overall relatives errors are observed for the first strategy.

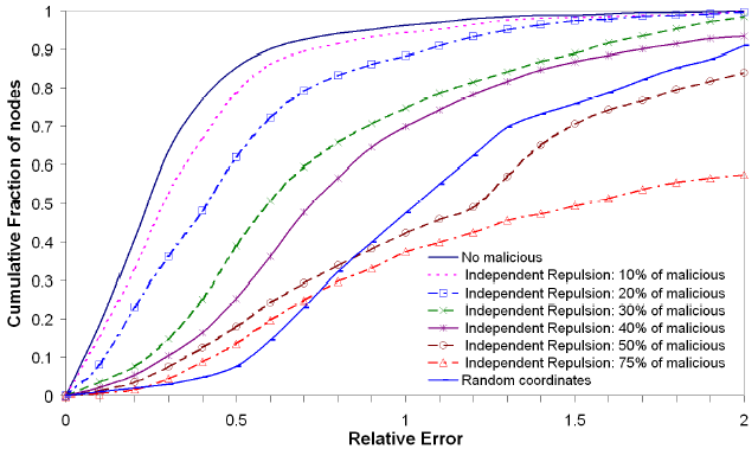


Figure 3.5: Repulsion attack on Vivaldi: CDF of relative error after 5000 ticks [4]

System control

The *System control attack* is possible on coordinate systems which allow any node to become a landmark. The principle is to take the control of a node which influence the coordinates of many other nodes or to become such a node. Thus, having the control of few nodes permits to dramatically affect the performances of the whole system.

Hierarchical systems like NPS are prime targets for this attack because the higher the node is in the hierarchy, the more the influence. On the contrary, a fully decentralized system like Vivaldi seems less interesting if the neighbourhood is well distributed among the nodes. Indeed, a particular node should not have more influence than another.

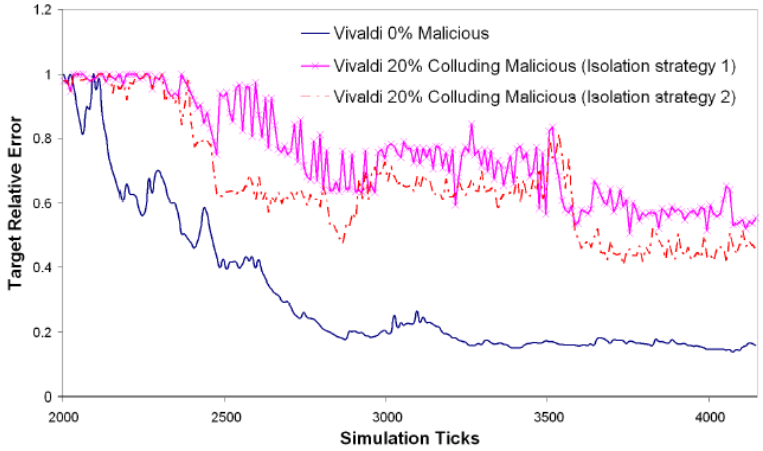
Combined attacks

It is reasonable to suppose that a large-scale always-on coordinates service has permanently few misbehaving nodes. Actually, after an outbreak, a small portion of nodes may not be corrected for a long time. As a result, the system may be targeted by different attacks at the same time. This type of attack is called a *combined attack*.

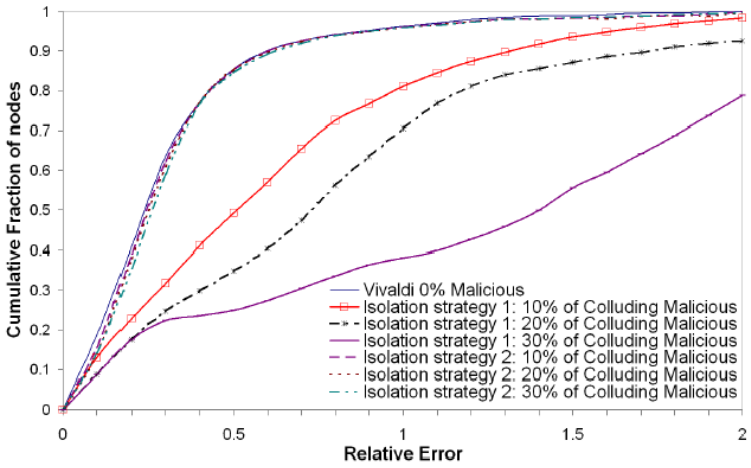
As we can see in the following, a fairly low level of malicious nodes conducting a combined attack can still have an impact on the whole system.

Combined attacks on Vivaldi: Fig. 3.7 depicts the impact of combined attacks on Vivaldi. This attack combines the attacks we have seen before and every attack is performed by the same number of attackers. The colluding isolation attack uses the first strategy.

This figure shows that even with few malicious nodes, it is possible to have an important impact on the accuracy of the system. Worst, this illustration indicates that the recovery after an attack can take a terribly long time.



(a) Evolution of relative error of the target nodes [4]



(b) CDF of relative error after 5000 ticks [4]

Figure 3.6: Analysis of colluding isolation attacks on Vivaldi

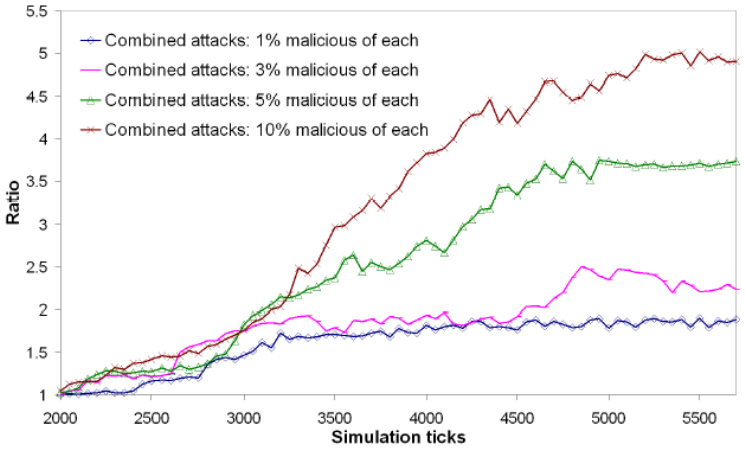


Figure 3.7: Combined attacks on Vivaldi: [4]

Kaafar et al. have also study the internal attacks on NPS. However, we choose not to present the attacks on NPS because they do not introduce new useful concepts for the rest of this work. Nonetheless, we invite the reader to consult [4] for the details on NPS and [16] for the details about attacks on Vivaldi.

3.4 Conclusion

In this chapter we have seen that coordinate system are sensible to attacks. These attacks can come from nodes outside the system but can also come from trusted nodes inside the system. For internal attacks, disorder attacks have a limited impact on the accuracy of the predicted distances compared to repulsion attacks. Indeed, while disorder attacks have no particular objective, repulsion attacks are the result of a smart analysis to move the nodes towards a defined position. The combination of different kind of attacks leads to worst coordinates than the choice of a single sort of attack.

In the next chapter, we see how to improve the robustness of NCS with a reputation based approach. We first formalize the securing procedure. After, we evaluate the real improvement of the proposed solution.

4 Protecting Coordinate Systems

In Chapter 3 we showed that NCS are sensible to attacks. The first attacks we presented are coming from the outside. The second type of attacks is more insidious and come from the system itself.

On one hand, good general techniques have be given to protect a system against external attackers (i.e., *hard* security like passwords, cryptography, certificates, etc.). On the other hand, no real solution has been proposed to protect a system from nodes that are inside the system. Indeed, it is hard to protect a system from nodes that are part of it and provide bad information. The analogy with the human world could be the following: It is simple to avoid someone to speak but hard to avoid him to lie. The inside attackers are detected with *soft* security like reputation [28, 29].

In this chapter we propose a reputation system that allows nodes to decide whether another node lies abouts its coordinates or not. We first introduce the concepts of *experience*, *trust* and *reputation* in Sec. 4.1. Next, we present the *uncertain probabilities model* and its specific operators that fit well with the concept of reputation in Sec. 4.1. After, we present a reputation model for embedding systems in general in Sec. 4.2 and apply it to Vivaldi in Sec. 4.2.4. Finally, we propose a cryptography and authentication layer on the model to resists to external attacks and to ensure privacy in Sec. 4.2.7.

4.1 Theoretical background

Before using the concepts of trust, experience and reputation, we have to define them precisely with definitions presented by Kinateder et al. [30]. In the following A, B and C indicate arbitrary nodes in the system.

Definition 1 (Experience) *An experience is an observation of A about some behavior of entity B.*

Definition 2 (Trust) *Trust is a subjective expectation an entity has about another's future behavior based on the history of their encounters.*
Trust is transitive, thus if A trusts B and B trusts C then A trusts C.

Definition 3 (Reputation) *The reputation of an entity A is the average (whatever average means in that context) trust of all other entities towards A.*

This definition of reputation suggests that reputation is global and objective when trust is local and subjective.

Jøsang proposes the *uncertain probabilities model* that gives interesting operators for reputation and trust computation [31, 32]. This model is based on three fundamental functions, the *belief* $b(x)$, the *disbelief* $d(x)$ and the *uncertainty* $u(x)$. Where $b(x)$ is the total belief an observer has that x state is true, $d(x)$ that it is not true and $u(x)$ expresses the uncertainty about x. We are encouraged to use such a model because the trust is based on the experience that is uncertain by nature.

With this theory, Jøsang demonstrates the theorem of belief function additivity in [31] that says:

$$b(x) + d(x) + u(x) = 1, \qquad x \in 2^{\Theta}, x \neq \emptyset. \tag{4.1}$$

Eqn.4.1 permits to understand that the uncertainty fills the gap in absence of belief and disbelief. If the observation x is certain (i.e., $b(x) = 1$), there is no uncertainty. On the contrary, if both the belief and the disbelief are small the uncertainty is important (i.e., if $b(x) = 0.1$ and $d(x) = 0.2$ then $u(x) = 0.7$). In other words, an important uncertainty indicates that the knowledge about x is limited.

The *opinion* w_x on x is a 3–dimensional metric that completely defines the uncertain probability on x [32]. The opinion the entity A has on x is defined as follows:

$$w_x^A \equiv \left(b_x^A, d_x^A, u_x^A\right). \tag{4.2}$$

where b_x^A is the belief A has on x, d_x^A is the disbelief A has on x and u_x^A is the uncertainty A has on x. All the possible opinions form the *opinion space*.

The uncertain probabilities model proposes two evidential operators: the *discounting* operator $\bigotimes$ and the *consensus* operator $\bigoplus$.

Definition 4 (Discounting operator $\bigotimes$) *If A has opinion $\omega_B^A = (b_B^A, d_B^A, u_B^A)$ on B and B has opinion $\omega_x^B = (b_x^B, d_x^B, u_x^B)$ on x, then A has the opinion $\omega_x^{AB} \equiv \omega_B^A \otimes \omega_x^B = (b_x^{AB}, d_x^{AB}, u_x^{AB})$ on x such that:*

$$\begin{aligned} b_x^{AB} &= b_B^A b_x^B \\ d_x^{AB} &= b_B^A d_x^B \\ u_x^{AB} &= d_B^A + u_B^A + b_B^A u_x^B. \end{aligned} \tag{4.3}$$

The discounting operator permits to have an opinion about a state x without having to directly estimate it. It is only required to have an opinion about something that has an opinion about that state. The discounting operator can be understood as the transitivity operator in the opinion space, the result of a discounting of an opinion from A and an opinion from B about a state x is still an opinion. Intuitively, the resulting belief in x is the product of the two input beliefs. As a matter of fact, the resulting belief must represent the belief B has in state x weighted by the belief A has in B. The same observation can be given for the resulting

disbelief. This disbelief is the disbelief B has in x weighted by the importance (the belief) A gives to B. The uncertainty is such that the discounting respects the belief function additivity (Eqn. 4.1).

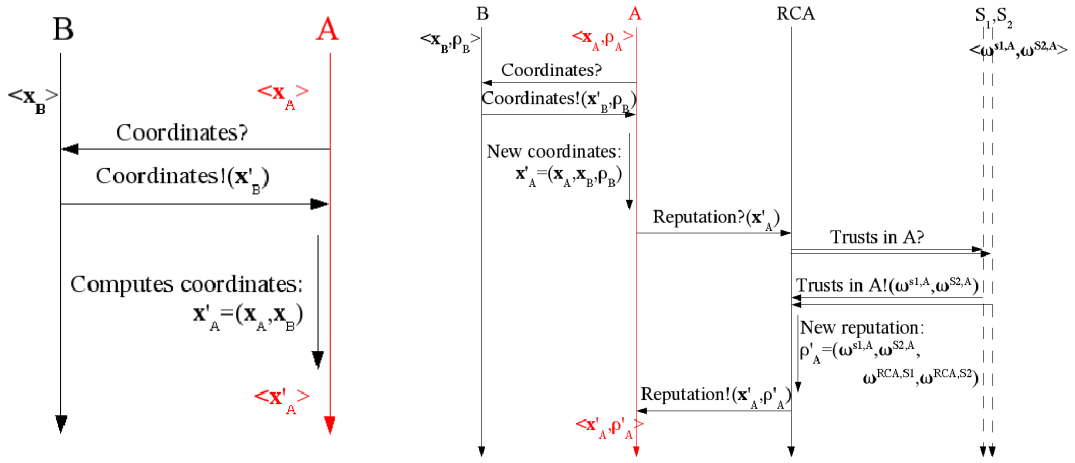
Definition 5 (Consensus operator $\bigoplus$) *For two different agents A and B, the opinions they have on x (ω_x^A and ω_x^B) may be different. To have a better estimate of the event's probability, the two observers may combine their observations and form a imaginary observer $[A, B]$ [31]. The consensus of $\omega_x^A = (b_x^A, d_x^A, u_x^A)$ and $\omega_x^B = (b_x^B, d_x^B, u_x^B)$ is $\omega_x^{A,B} \equiv \omega_x^A \oplus \omega_x^B = (b_x^{A,B}, d_x^{A,B}, u_x^{A,B})$ such that:*

$$\begin{array}{ll} b_x^{A,B} & = (b_x^A u_x^B + b_x^B u_x^A)/\kappa \\ d_x^{A,B} & = (d_x^A u_x^B + d_x^B u_x^A)/\kappa \\ u_x^{A,B} & = (u_x^A u_x^B)/\kappa \end{array} \tag{4.4}$$

where $\kappa = u_x^A + u_x^B - u_x^A \cdot u_x^B$

The consensus operator permits to define a sort of average opinion multiple entities have in a single observation. The result of the consensus of two opinions is an opinion. The consensus operator permits to decide an opinion on a state x based on the opinion on x from A and from B. The resulting belief is the sum of the beliefs weighted by the uncertainties associated to these beliefs. The disbelief is computed in the same way and the resulting uncertainty is the product of the uncertainties. Belief, disbelief and uncertainty is normalized by a factor κ to respect the belief function additivity (Eqn. 4.1).

Jøsang has demonstrated $\otimes$ is associative but not commutative and $\oplus$ is commutative and associative [31]. Thus, the order in which opinions are combined has no importance.



(a) Classic embedding system interactions

(b) Reputation-based embedding system interactions

Figure 4.1: Interactions between nodes

4.2 A Reputation Model for NCS

We saw in Sec. 3.3.2 that embedding systems are sensible to internal attacks. A solution for making embedding systems robust to internal attack is to use reputation. In this section, we propose our reputation model for NCS in general.

In traditional NCS (Fig. 4.1(a)), any node A updates its coordinates based on the coordinates of one of its neighbours and the distance to it. In our new approach (Fig. 4.1(b)), the new coordinates also depend on the reputation of the neighbours. When A updates its coordinates based on measurements with neighbour B, it first contacts B to retrieve its coordinates and reputation. A then computes its coordinates as a function of its own coordinates, B's coordinates and B's reputation. Then, A contacts a special certification agent, the *Reputation Computation Agent* (RCA) to update its own reputation. This RCA is similar to the RCA proposed by [33]. The RCA is used to construct a reliable reputation for any node in the embedded system. For this, we follow the recently proposed approach by Kaafar et al. [34] and introduce new entities in the system: The *surveyors*. Some surveyors are attached to each node in the system. Surveyors are well chosen nodes that perform experiences measurements and trust estimation on other nodes. The number of surveyors depends on the number of malicious nodes in the system. Next, the RCA computes its own trust to A's surveyors. Finally, the RCA computes the new reputation of A with all these parameters.

In the following, we propose a more formal approach to the notions of experience, trust and reputation. We also define the behavior of the two concepts of RCA and surveyors. We first presents a model for the experience. After, we present a model for the trust. With these two concepts, we construct a formal reputation model and a model to rank the nodes based on the reputation. We then propose a solution to protect the information and to choose the surveyors. Finally, we modify the Vivaldi algorithm to include our reputation model.

4.2.1 Experience Model

At time t, an experience is an observation of a node A about some behavior of another node B. This observation is evaluated as follows:

$$\xi(A,B,t) = 1 - \frac{\left|\hat{d}(A,B,t) - d(A,B,t)\right|}{\max\left(d(A,B,t), \hat{d}(A,B,t)\right)}. \tag{4.5}$$

Where $\hat{d}(A,B,t)$ is the estimated distance between A and B and $d(A,B,t)$ is the real distance. The relative error gives information about the accuracy of the predicted distances. The lower the relatives errors are, the accurate the coordinates are. The experience converts the relative error in the bounded interval $[0,1]$. The experience is maximum for a perfect estimation and decreases with the augmentation of the *prediction error*.

4.2.2 Trust Model

The trust A has in B is an expectation of the future behavior of a node based on the previous experiences A has in B. However, the experience we defined before depends on external elements and is inherently not absolutely reliable. The concept of uncertain probabilities explained in section 4.1 permits to model this doubt.

Conceptually, the trust must limit the risk of multiple identities. It incites therefore nodes to remain in the system for a long time. However, the trust must stay enough reactive to adapt to sudden changes in the topology [33]. The introduction of the concept of *trustworthiness* permits to reach these two opposite requirements.

The trustworthiness $\tau(A,B,t)$ of A in B at time t is an exponential averaged sum of the experiences [32] multiplied by an *ageing factor*:

$$\tau(A,B,t) = a(t) \cdot \gamma \cdot \left(\sum_{i=0}^{h} (1-\gamma)^i \cdot \xi(A,B,t-i) \right). \tag{4.6}$$

where $a(t)$ is the ageing factor ($a(0) = 0$), γ is a weighting constant and h is the number of previous experiences that must be taken into account. The exponential averaged sum of the experience permits to give more importance to the most recent experiences [35]. The ageing factor increases with the seniority and limit the trustworthiness of recent nodes (similar to the loss factor proposed in [11]). It is defined as follows:

$$a(t) = c_a + (1 - c_a) \cdot a(t-1). \tag{4.7}$$

Where c_a is the *age bonus coefficient* such that $0 < c_a < 1$ and $a(0) = 0$. c_a controls the gain of the age for the trust computation. The value of c_a is a tradeoff between wisdom and convergence time. A low value of c_a implies a slow convergence to 1, meaning that only old

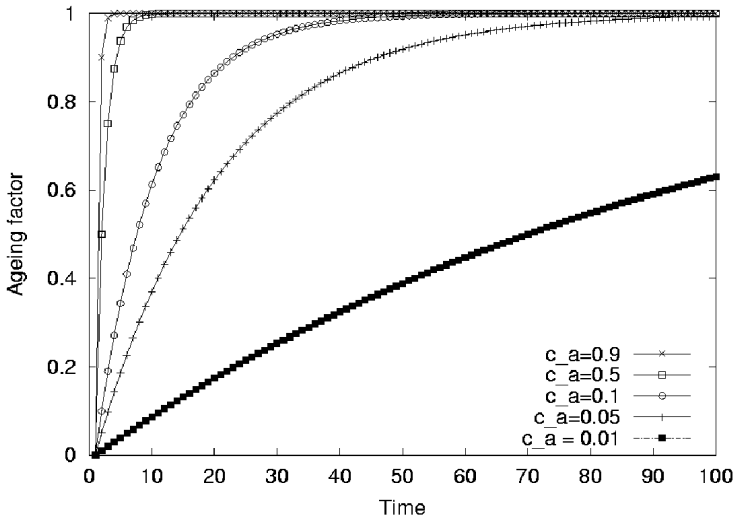


Figure 4.2: Evolution of the ageing factor for different c_a.

nodes may benefit completely from the experiences. On the contrary, a large value for that factor quickly rises the ageing factor to 1 leading recent nodes to use their entire experience rapidly. Fig. 4.2 depicts the evolution of the ageing factor with the computation cycles for different values of c_a ($c_a = 0.01$, $c_a = 0.05$, $c_a = 0.1$, $c_a = 0.5$ and $c_a = 0.9$). The ageing factor is a function that always converges to 1 more or less rapidly depending on c_a. In Fig 4.2, we see that the more the age bonus coefficient is, the faster the convergence of the ageing factor is. For $c_a = 0.9$, the trustworthiness of a node is totally used after less than 5 computations. For a small $c_a = 0.01$, the trust is less than the half of the averaged sum until 70 computations. For this c_a, the averaged sum is used by less that 65% after 100 iterations. It is interesting to see that for small values of c_a, the ageing factor is quasi-linear.

The *untrustworthiness*, $\bar{\tau}(A, B, t)$, is the complement to 1 of the trustworthiness:

$$\bar{\tau}(A, B, t) = 1 - \tau(A, B, t). \tag{4.8}$$

The *doubt* $\varepsilon(A, B, t)$ A has in B at time t is the variation of the experiences with the time. This variation is estimated with the variance of the last h experiences:

$$\varepsilon(A, B, t) = \sigma^2 \left(\bigcup_{i \in \{0..h\}} \xi\left(A, B, t - i\right) \right). \tag{4.9}$$

where $\sigma^2(\cdot)$ is statistical variance with $N - 1$ degrees of freedom.

The model of uncertain probabilities offers strong perspectives to the reputation in general. The trust $\omega(A, B, t) = (b_B^A(t), d_B^A(t), u_B^A(t))$ the node A has in B at time t has the following bijection with the 3 concepts of trustworthiness, untrustworthiness and doubt:

$$\begin{aligned} b_B^A(t) &= \frac{\tau(A,B,t)}{(\tau(A,B,t)+\bar{\tau}(A,B,t)+\varepsilon(A,B,t))} \\ d_B^A(t) &= \frac{\bar{\tau}(A,B,t)}{(\tau(A,B,t)+\bar{\tau}(A,B,t)+\varepsilon(A,B,t))} \\ u_B^A(t) &= \frac{\varepsilon(A,B,t)}{(\tau(A,B,t)+\bar{\tau}(A,B,t)+\varepsilon(A,B,t))}. \end{aligned} \tag{4.10}$$

The common denominator in Eqn. 4.10 normalizes the sum of trustworthiness, untrustworthiness and doubt to 1. Thus, the trust respects the belief function additivity (see Eqn. 4.1) and is in the opinion space.

4.2.3 Reputation Model

The reputation of an entity at a particular time must be unique and must be function of the trust all nodes have in it. However, for scalability reasons, it is impossible to construct a full-mesh reputation model in which each node cooperates with all others to share trust information. We therefore propose a *pseudo-reputation* model in which only a few nodes cooperate to evaluate the reputation.

The reputation model needs the introduction of two new types of nodes: the *Reputation Computation Agent* and the *surveyors*.

Definition 6 (Reputation Computation Agent or RCA) *The RCA is a well-known special node which purpose is to compute the pseudo-reputation of every nodes of the system. The RCA does not have coordinates, is always on and is accurate [33].*

Definition 7 (Surveyors) *Surveyors are entities that periodically compute trust to other [34]. Every node A has a set $\mathcal{S}_A$ of n surveyors associated to it ($RCA \notin \mathcal{S}_A$).*

Every node has a set of well-chosen surveyors assigned to it by the RCA. The surveyors are normal nodes in the system. A surveyor periodically performs experience measurements on its set of assigned nodes. When the reputation of a node A has to be updated, the RCA computes its trust in the A's surveyors and combines these trusts with the trust the surveyors have in A. This process is formalized as follows:

$$\hat{\omega}_A^{RCA} = \bigoplus_{\{H_n \in \mathcal{S}_A\}} \tilde{\omega}_{H_n}^{RCA} \otimes \hat{\omega}_A^{H_n}. \tag{4.11}$$

Where $\tilde{\omega}_{H_n}^{RCA}$ is the opinion RCA has in H_n, the n^{th} surveyor of A. In the opinions computed by the RCA, the experience is not computed with Eqn. 4.5 but with Eqn. 4.12. This particular experience is introduced to avoid the RCA to have coordinates. Indeed, if the RCA had coordinates, it would be easy for an attacker to alter the coordinates of the RCA and invalidate the reputation model. The experience as computed by the RCA is presented in Eqn. 4.12.

$$\xi(RCA, H_n, t) = 1 - \frac{\sqrt{\sigma\left(\overrightarrow{v}_A^{H_n}(t)/n^2\right)}}{\#\overrightarrow{v}_A^{H_n}(t)}. \tag{4.12}$$

where $\overrightarrow{v}_A^{H_n}(t)$ is the *variation history vector*, $\#\overrightarrow{x}$ is the size of the variation history vector, n is a normalization factor and $\sigma(\cdot)$ is the *variance* of the set. The experience with H_n computed by the RCA is an indicator of the variation of the coordinates of H_n normalized to 1. If a node has important changes in its coordinates, its experience is low (an entity which always change its vision of the world is not reliable). On the contrary, if only few coordinates changes are observed, the experience is better and the node is considered as more reliable. In Eqn. 4.12, the intuition behind the division by $\#\overrightarrow{v}_A^{H_n}(t)$ is that a large variance in a small set is more abnormal than a similar variance for a large set. The normalization permits to bound the experience in $[0, 1]$.

The variation history vector is the history at time t of the last h variations of coordinates RCA has observed for node A. The variation history vector (*VHV* for short) is defined as follow:

$$\overrightarrow{v}_A^{H_n}(t) = \langle \|\overrightarrow{c}_{t-h} - \overrightarrow{c}_{t-h+1}\|, \ldots, \|\overrightarrow{c}_{t-1} - \overrightarrow{c}_t\| \rangle . \tag{4.13}$$

The normalization factor n is presented in Eqn. 4.14 and is the maximum value of the experience that the RCA has observed for the considered node H_n. In other words, n is the maximum of the experiences in the VHV of all the surveyors of H_n.

$$n = argmax \left(\bigcup_{\{H_n \in \mathcal{S}_A\}} \overrightarrow{v}_A^{H_n}(t) \right) . \tag{4.14}$$

A node must update its reputation in two cases: (i) the node has changed its coordinates or (ii) the reputation has expired. The first case limits the possibility for a malicious node to change its coordinates all the time. Actually, variations in coordinates increase the doubts the different surveyors have in the node. High doubts imply low trusts and thus bad reputation. The RCA may also protect the system by limiting the number of changes per time unit. Second, the reputation is limited in time because coordinates may change with the time and an accurate coordinates at a given time may be inaccurate later. We suppose each node synchronizes a coarse-grained clock with the RCA's clock using NTP [36]. The accuracy of this clock is not too important and an error of several seconds might be acceptable. The expiration date may be adaptative to propose short periods at the start of the node and longer periods when coordinates are stabilized. With the expiration date, the reputation may be seen as a lease.

For many applications, it would be more useful to have a *scalar reputation* for A instead of the vectorial reputation $\hat{\omega}_A^{RCA}$. We propose the scalar reputation $\hat{\varrho}_A$ based on the opinion $\hat{\omega}_A^{RCA}$ in the equation 4.15.

$$\hat{\varrho}_A = \hat{b}_A^{RCA} \cdot (1 - \hat{u}_A^{RCA}). \tag{4.15}$$

The scalar reputation is the belief in A weighted by the uncertainty that persists on that affirmation and is bound in $[0, 1]$. We have to notice that if the uncertainty is total (i.e., $\hat{u}_A^{RCA} = 1$), the resulting reputation is 0. If the uncertainty is minimum (i.e., $\hat{u}_A^{RCA} = 0$), the reputation equals the belief $\hat{b}_A^{RCA}$.

The reputation can be used during the computation of coordinates. The reputation is the probability the neighbour is reliable, if the neighbour has a high probability, the node can update its coordinates with all the information given by its neighbour. However, if the probability is low, the update must limit the impact of the neighbour information for the new coordinates. The idea is general and must be adapted for every coordinate system. Section 4.2.4 illustrates it in Vivaldi.

4.2.4 Application to Vivaldi: RVivaldi

In the previous section, we have seen how to secure a coordinate system. In this section, we adapt Vivaldi to embed the reputation. This adaptation of Vivaldi is called *Reputation-Based Vivaldi* (*RVivaldi*).

The principle of RVivaldi is to modify the adaptative timestep δ of Vivaldi to take into account the reputation. The timestep defines the quantity of allowed displacement of the node due to the force exerted by the spring connecting it to its neighbour. When a node is perfectly reliable, its reputation is set to 1. On the contrary, if a node is absolutely not reliable, its reputation is set to 0. The reputation can be seen as the probability that the node is reliable and, if the reputation multiplies the timestep, which is also a probability ($\delta \in [0,1]$), the displacement of the node is weighted by the reputation.

Vivaldi has been proposed for environments without attackers and works well in that case. The idea is to keep using Vivaldi when the neighbour is reliable and to limit the modification of coordinates proportionally with the reliability of the neighbour.

RVivaldi only requires to modify Eqn. 2.14 of Vivaldi. Eqn. 2.14 computes the new coordinates of B based on its observation of A. The result is presented in Eqn. 4.16.

$$\overrightarrow{x}_B = \overrightarrow{x}_B + (\hat{\varrho}_{\mathbf{B}} \cdot \delta) \cdot \left(d_{BA} - \hat{d}_{BA}\right) \cdot u\left(\overrightarrow{x}_B - \overrightarrow{x}_A\right) \tag{4.16}$$

4.2.5 Ranking model

Coordinates are often used to determine whether B or C is the closest node to a node A. A solution to choose the closest node is to *rank* the choices in order of preference. To build the rank, we propose the *score*, an indicator that permits to classify nodes according to the distances and reputation.

If the reputation is not taken into account, the node with the smaller distance can be selected as the closest node. The reputation may improve the quality of the decision: Not choosing the nearest node but the nearest node with the better reputation. If a node with a really bad reputation is a little bit closer than a node with a high reputation, it may be good to choose the node we are more confident in. This decision is an observation of real life: You might probably accept to pay more to have a good technician than pay less and have a bad guy.

The idea presented above is formalized by the score. The higher the score, the more attractive the node. The score must classify the nodes as a tradeoff between the distance and

the reputation. For two nodes at the same distance, the score must be higher for the node with the better reputation. In the same manner, for two nodes with the same reputation, the closest node must have the better score. Nevertheless, the reputation prevails over the distance. Indeed, a node with a large distance but better reputation may have a better score than a node with a smaller distance but a worst reputation. The score ς_B^A that A computes for node B is defined in Eqn. 4.17.

$$\varsigma_B^A = \frac{1}{\hat{d}_{AB} + (1 - \hat{\varrho}_B) \cdot 1} \cdot \hat{\varrho}_B. \tag{4.17}$$

The score is computed in two parts and must fall within the range $[0, \infty]$. First, a low value is affected to long distances. Second, a bad reputation is sanctioned. The result is that for two nodes at the same distance, the node with the higher reputation has a better score. In any case, the more interesting node is the one with the highest score. In Eqn. 4.17, the complement of the reputation multiplied by the unitary distance is added to the distance in the inverse relation to avoid nodes with bad reputation to have the maximum score[1]. The maximum score can only be reached for a node at the same position as the asking node and with a perfect reputation of 1. Without the additional term in the denominator, the score might be equal for nodes at the same position but with different reputations when the distance is zero. This is not acceptable. The inverse of the function of the predicted distance is used to give a better score to nodes that are close.

Fig. 4.3 depicts the evolution of the score with the distance and the reputation. The x-axis represents the distance (in the interval $[0.1, 1280]$), the y-axis is the reputation (in the interval $[0, 1]$) and the vertical axis is the reputation. As expected, the score increases for small distances and high scores. For distances bigger than 1, the score is always lower than 1 and rapidly converges to 0. On the contrary, when the distance is in $[0, 1[$ the score can be higher than 1 and a small difference in the distance or reputation causes an important change in the score. This observation incites us to propose to norm, when calculating the score, all the measured distances in $[0, 1]$, by dividing by the higher predicted distance, for instance. For a perfect reputation of 1, the score is the inverse of the distance. For the worst reputation (i.e., a reputation of 0), the score is always 0. Fig. 4.3, we can observe that the score looks constant for important distances but it slightly tend to 0 when the reputation decreases and the distance increases.

To summarize, when node A must decide whether of B and C is the nearest it first computes ς_B^A and ς_C^A. After, A construct a ranked list where the nodes are placed ordered by the value of the score. The closer node is the node with the higher score in the list[2]. The proposed solution is defined for nearest (in distance) node selection and the technique must be adapted for other usages.

4.2.6 Choice of surveyors

The RCA is responsible for the choice of surveyors for every node. This choice is strategic for the reputation model. Indeed, the surveyors must be chosen such that their probability to be malicious and to collude for an attack is as low as possible. In addition, the set of surveyors must be representative of the whole system. Moreover, it must be impossible for a

[1] $1 - \hat{\varrho}_B$ is multiplied by the unitary distance 1 to remain consistent with the dimension of $\hat{d}_{AB}$.
[2] The node with the most important distance is the one with the lower score.

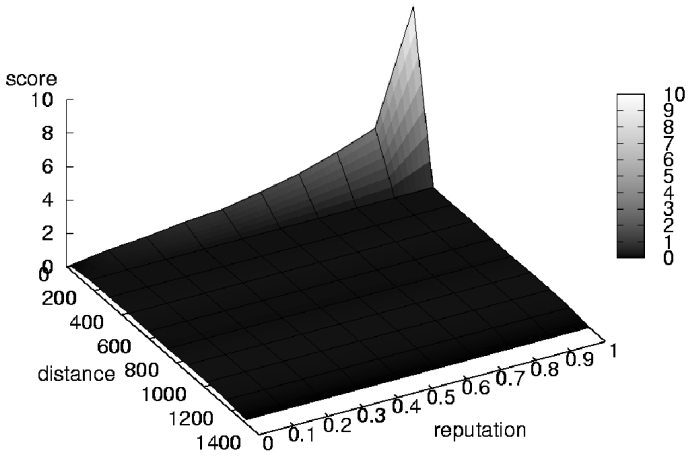


Figure 4.3: Evolution of the score as a function of the reputation and the distance

node to know the surveyors of other nodes (see Sec. 4.2.7). This last point is required to avoid a malicious node to perform surveyor's specific attacks.

If we randomly choose the surveyors, the probability of having a malicious surveyor should be the same as the probability of having a malicious node in the system. The random selection of surveyors is good but can be improved: The surveyors are still randomly chosen but a particular property must be maximized. The property we propose to maximize is the distance between IP addresses in a set of surveyors. Such a solution permits to be confident in the fact that the surveyors are well distributed over the Internet because their IP addresses are strongly different. A solution to maximize the distance between IP addresses is to maximize their *Hamming distance*:

Definition 8 (Hamming distance) *Let $\bar{x}$ and $\bar{y}$ be two binary n-uples $\bar{x} = (x_1, \ldots, x_n)$ and $\bar{y} = (y_1, \ldots, y_n)$. The Hamming distance $h(\bar{x}, \bar{y})$ between $\bar{x}$ and $\bar{y}$ is:*

$$h(\bar{x}, \bar{y}) = \{i \in [1, n] : x_i \neq y_i\} \tag{4.18}$$

In other words, the Hamming distance is the number of positions in two binary words for which the symbols are different [37]. For example, the Hamming distance between 6 and 5 is 2 because the binary representation of 6 is **110** and for 5 we have **101**. IP addresses are binary words with a length of 32 bits in IPv4 and 128 bits in IPv6.

The set $\mathcal{S}_A$ of size n of surveyors is constructed such that the distances between any pair of two IP addresses in the set is maximized. The subset is such that the median of the Hamming distances is as high as possible. The median of the Hamming distances permit to see if the surveyors are globally well distributed, which should not be seen with the average. Another solution could be to solve an optimization problem on the distances but it is recommended to have light algorithms to limit the risk of DoS by overload.

In practice, the RCA randomly chooses a set $\mathcal{S}$ of nodes among the running nodes and infers a subset $\mathcal{S}_A$ of size n from it such that $\mathcal{S}_A \subseteq \mathcal{S}$. Some criteria may be added for the

construction of $\mathcal{S}$. For example, the selection may be performed such that the set is always composed of nodes that must survey a minimum number of nodes. Thus, the load on surveyors is limited and uniformly balanced.

One could maximize the Hamming distance between the most significant bits of the IP addresses instead of maximizing the Hamming distance of the entire IP address.

Instead of maximizing the Hamming distance, we could choose the surveyors such that their coordinates are well-spread in the virtual space and choose the nodes with the better score. This solution is less robust than the selection based on IP addresses. Indeed, it is easy for malicious nodes to choose their own coordinates (but the resulting reputation is poor). On the contrary, it is difficult for an attacker to take the control of nodes in many different sub-networks. In addition, using the coordinates and the reputation to choose the surveyors lead to a conflict of interests.

We propose in Sec. 5.5 a methodology to evaluate the best number of surveyors and RCA, depending on the number of attackers.

4.2.7 Protection and privacy

The robustness of the reputation model relies on the fact that it is impossible for a node to know which node is a surveyor of another. Indeed, if a malicious node was able to determine the surveyors of its target, it could attack all its surveyors and decrease its attractiveness. In addition, if the messages are not protected, it is possible for an external attacker to tamper their content or to inject fake messages. In this section, we propose how to protect messages against tampering and eavesdropping.

The scalar reputation is sent by the RCA to the node as a ticket. So that, when a node has to use the coordinates of A, it just has to contact A and retrieves the ticket. Hence, the RCA is never contacted for a coordinates retrieval and does not become a bottleneck [33]. When a node gets the coordinates of A, it also receives $\hat{\varrho}_A$. However, to be sure the reputation has not been tampered by A or any other attacker, the reputation must always be signed by the RCA. Because of the protection against tampering, every message must be signed by the sender. We propose to use *Cryptographically Generated Addresses* (CGA) to sign messages without having to rely on a specific infrastructure [38]. Some bits of an IPv6 cryptographically generated addresses are generated by hashing the public key of the address owner. To sign a message, the address owner uses the corresponding private key to assert the address ownership.

To avoid repetition of a message, a *nonce* η is added in every message. A nonce is a number to is never used more than one time. When a nonce is used twice, the second message with the same nonce must be dropped. The nonce permits to avoid attacks by repetition.

Unfortunately, the signature only protects against tampering. But we have seen before that some parts of the message can contain valuable information for attacks. Thus, critical information must be encrypted such that only the target of the message is able to decrypt it.

To interact in the system, the node must have a valid *ticket*. The ticket sent by the RCA contains the reputation and a validity period. The reputation contains in the ticket gives the capacity of the node to interact with the system. This ticket presents a limitation in time because the topology of a network might change all the time and a correct node at a given

time may become a bad one. When the ticket expires, the node must ask the RCA for a new ticket. The lifetime of the ticket depends on the network behaviour. A stable network can use long lifetime and, on the contrary, a network of mobile nodes should use short times.

The messages exchanged between surveyors and the RCA may contains important information for attackers. Fortunately, surveyors are normal nodes and must periodically refresh their tickets like any other node. This property gives the ability to hide the transfer of the results of the surveys to the surveyors by pickibagging them on the request for the ticket. This avoid the creation of specific messages for surveyors and thus limit the risk of detection by malicious nodes.

When a node A asks the RCA its current reputation, it follows the following steps. First, A sends a ticket request to the RCA. Then, The RCA answers with a challenge request. Third, A sends its coordinates, variation history vector and the list of all opinions it has in all the node it is the surveyor for and the challenge response (i.e., a response to the challenge request). Finally, if the challenge response is correct, the RCA computes A's reputation, sends it back and stores the last information for the surveyor part of the node.

The detail of this protocol is presented in Fig. 4.4.

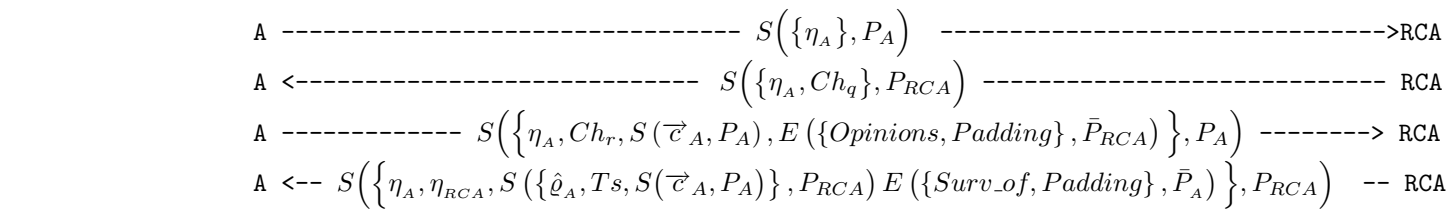


Figure 4.4: Secured reputation protocol

In the messages of Fig. 4.4, η_X is a nonce generated by X. $S(m, P_X)$ is the message m signed by X using its private key P_X. $E(m, \bar{P}_X)$ is the message m encrypted to be readable only by X using its public key $\bar{P}_X$. T_s is the timestamp (the expiration date) assigned to the reputation. Actually, after several time, the reputation may have changed and an up-to-date value must be retrieved.

The nonce η_A in the ticket request is an identifier for the request. All the messages relative to this request must contain this number as a proof that the message is an answer for that request. This solution permits to limit the risk of repetition attacks.

The challenge introduction mitigates the impact of DDoS attacks. Actually, the main overload of the RCA occurs during the reputation computation (i.e., decrypt, compute opinions, encrypt, etc.). The challenge permits to reduce the number of reputation computation by imposing the malicious nodes to have a bidirectional communication with the RCA. Actually, the challenge imposes the attacker to provide a valid address and to be able to compute the challenge response to continue the attack. The RCA sends the challenge query Ch_q and the node answers with Ch_r, the challenge response. The challenge must be easy to compute and to verify by the RCA. On the contrary, Ch_r must be hard enough to compute and thus dissuade a malicious node from flooding the RCA with reputation computations. The challenge query/response procedure can be implemented using a *network puzzle* [39]. For instance, Ch_q could be a nonce and Ch_r a string that has Ch_q as hash value. The RCA computes the reputation only if the challenge response is correct.

To limit the number of exchanged messages, the challenge response also contains the current coordinates of the node. The answer also contains the list of the opinions the node has about the hosts it is a surveyor for. The size of the list is an indication of the number of nodes the sender is a surveyor for. This is why a padding is added at the end of the list such that the size of the list is always the same whatever the number of surveyed nodes. The list of opinions is sent encrypted to be impossible for an attacker to detect the entities surveyed by the node. Again, the size of the encrypted part of the message must always be of the same size. This length depends on the maximal number of nodes a surveyor is responsible for. Some other precautions must also be taken to be sure that it is impossible for an attacker to detect the real size of the list, for example by measuring the time taken by the node or the RCA to compute the encrypted list. In addition, it is important to keep opinions anonymous to avoid a poorly reputed node to punish the surveyors which give it a bad opinion. When the RCA receives the list of opinions, it stores it. The opinions in the list are used when the RCA has to compute the reputation of a node surveyed by the node that has sent the list.

When a node B asks A its coordinates, it sends A a nonce η_B and gets back a message with the coordinates, the reputation and the nonce:

$$S\Big(\{\eta_B, S\left(\{\hat{\varrho}_A, Ts, S(\overrightarrow{c}_A, P_A)\}, P_{RCA}\right)\}, P_A\Big).$$

When B receives the answer with the nonce corresponding to the sent nonce, it extracts A's coordinates and reputation. Then, B measures the RTT with A using ping or any technique to measure RTT.

The opinion a surveyor has about an attached node is based on its RTT and coordinates. To construct accurate opinions, a surveyor periodically measures the attached nodes. Similarly, when a node computes its coordinates, it periodically contacts its neighbours to retrieve their coordinates and RTT. This property permits to hide the surveyors in the exchange of messages. Indeed, if a surveyor behaves with its attached node like a normal node with its neighbours, it is impossible for an attacker to decide whether the message is from a surveyor or from a normal node updating its coordinates since the interesting content of messages is only accessible to the RCA.

The RCA can change the surveyors from time to time and can change their number. When the RCA decides to make S a surveyor of a node A, it must not contact S immediately to ask it to survey A. The RCA informs S to survey A only when S requests something to the RCA. This solution permits to avoid A or any other node to detect that S is a surveyor. This information is sent to S in the $Surv\_of$ part of the fourth message in our protocol. For the same reason as the opinion list, this information must be encrypted and of constant size. This artifact delays the first measurement for the reputation by a time equals to the double of the maximum lifetime of a ticket. Indeed, S queries the RCA at least one time during this period and the result of the first measurement arrives at the next ticket expiration.

The protocol is implemented on top of *TCP* and, if a node breaks down, the RCA detects it and reassigns a surveyor for the nodes surveyed by the removed entity. The RCA may also redistribute the surveyors at the addition of a node in the system such that the nodes always have a minimum set of nodes to survey.

4.3 Conclusion

In the previous chapter, we presented the different attacks the network coordinate systems are sensible to.

In this chapter, we proposed a reputation model to secure coordinate systems. In this model, the nodes are monitored by other nodes: The surveyors. When a node changes its coordinates, it contacts a special certification agent: The RCA. The RCA then computes how correct the coordinates are based on the observations made by the surveyors. This metric is evaluated by the reputation. The reputation is a value that indicates the probability that a node does not lie (i.e., a value near 0 means that the probability the node lies is important. On the contrary, a value near 1 means that the node seems to be honest). We propose to use the reputation during the computation of the coordinates. Indeed, if a node has a high reputation, one could use its information without particular attention. On the contrary, if a node has a low reputation, information given by the node must be taken with care. In addition to coordinates computation, the reputation can be used to select the closer node. To this aim, we proposed the score. In traditional network coordinate systems, the closer node is the one with the lower predicted distance. With the score, the distance is still minimized but nodes with a good reputation are favoured.

In this chapter, we proposed a new network coordinate system: RVivaldi. RVivaldi, is an adaptation of Vivaldi with the reputation model embedded. To secure Vivaldi with the reputation model, it is only required to modify the timestep function (δ) of Vivaldi. In Chapter 5, we evaluate the reputation model on RVivaldi to determine the level of protection the reputation model gives to coordinate systems.

5 Evaluation

Chapter 3 showed the different attacks that can affect coordinate systems. In Chapter 4 we propose a reputation model that makes coordinate systems robust to these attacks. In this chapter, we evaluate RVivaldi, the adaptation of Vivaldi that embeds the reputation model. This evaluation is the result of different simulations with the same data set as used by Dabek et al. in [2] when presenting Vivaldi and Kaafar et al. in [16] when they have presented several attacks on Vivaldi. This data set was construct with *King*, a tool that estimates latencies between nodes using active measurements on the existing DNS infrastructure [19].

5.1 Evaluation method

We learned in Chapter 3 that attacks on coordinate systems can be divided into internal and external attacks. In this evaluation, we only focus on internal attacks. We simulate four different attacks: (*i*) *random attack*, (*ii*) *constant attack*, (*iii*) *same attack* and (*iv*) *repulse attack*.

In a random attack, every time a malicious node is asked for its coordinates, it replies with a random one and a low constant value for its local error. On the contrary, in a constant attack, the malicious node always gives the same coordinates and the same local error randomly chosen at start up. In a same attack, a malicious node pretends its coordinates are the same as the coordinates of the node which requests the coordinates. It also gives a low constant local error. In the last attack, i.e., repulsion attack, a malicious node delays the round-trip time such that it looks consistent with a random coordinates it chooses for its target at the beginning. The malicious node wants to move the target to the decided coordinates. A repulse attacker tries to move all the nodes connected to it to the same randomly chosen target. All the repulse attackers use the same target coordinates.

Random and constant attacks are examples of disorder attacks. The same attack, for its part, is an example of a mix between disorder and repulsion attacks. Indeed, in same attack, the good nodes are repulsed from their coordinates but no intelligence is given to the displacement. The repulse attack is the example of repulsion attack given by Kaafar et al. [16].

For the experiments, every RVivaldi node has 32 neighbours as recommended by Vivaldi authors and 5 surveyors[1] all randomly chosen. Moreover, the Vivaldi tuning parameters are set to the recommended values [2]: $c_c = 0.25$ and $c_e = 0.25$. Our results are obtained for a 3-dimensions Euclidean space.

[1] The number of surveyors has been chosen arbitrarily.

Parameter	Values
γ	$\{0.01, 0.1, 0.2, 0.5\}$
h	$\{10, 20, 40\}$
c_a	$\{0.01, 0.1, 0.2, 0.4, 0.8\}$
% of malicious nodes	$\{0, 10, 20, 50, 70\}$
Attack	$\{random, constant, same, repulsion\}$

Table 5.1: Domain of the different parameters of the simulation

We are interested to see the improvement due to the reputation for the different attacks described above. However, the reputation depends on the three following parameters: The number of reminded experiences (h), the weighting constant γ and the age bonus coefficient (c_a). Thus, we first evaluate how the reputation evolves with different values of these constants and propose the best tradeoff. Moreover, the results should also depend on the attacks and the number of attackers. So that, the experiments have been performed for different values of the parameters but also with the different attacks and various number of malicious nodes. The possible values we choose for the parameters are presented in Table 5.1.

With such domains, we have no less than $4 \cdot 3 \cdot 5 \cdot 5 \cdot 4 = 1200$ possible combinations. The choice of these domains permits to cover a broad range of possibilities.

After, the performances of RVivaldi are compared with those of Vivaldi, using the best choice of parameters.

The attackers (i.e., the malicious nodes) are randomly chosen before the simulation begins and always perform a predefined attack during a simulation run. They perform their attack immediately when the simulation starts.

In the next sections, we evaluate the performance of our reputation model adapted to Vivaldi. After, we see how Vivaldi behaves with our attacks. Then, we decide the best choice of γ, h and c_a. Finally, we compare the performances of RVivaldi in their own and compared with Vivaldi.

5.2 The simulator

This section presents the simulator written in JAVA. The sources of the simulator are given in Appendix B.

Fig. 5.2 shows the class diagram of the most important classes of the simulator. The class `Lib` contains the simulator parameters and useful methods. The class `Node` implements the basic functionality of the nodes. The class `SurveyorNode` extends `Node` and implements the interface `Surveyor`. Any class that implements `Surveyor` can be used as a surveyor when needed in the simulation.

To implement a coordinate system algorithm, the class must extends `Node` and implements `Surveyor`. `VivaldiNode` is the implementation of the Vivaldi algorithm for the simulator. If algorithm must support the reputation, it must also implements the `ReputationNode` interface. `RVivaldiNode` implements RVivaldi, a NCS algorithm with reputation.

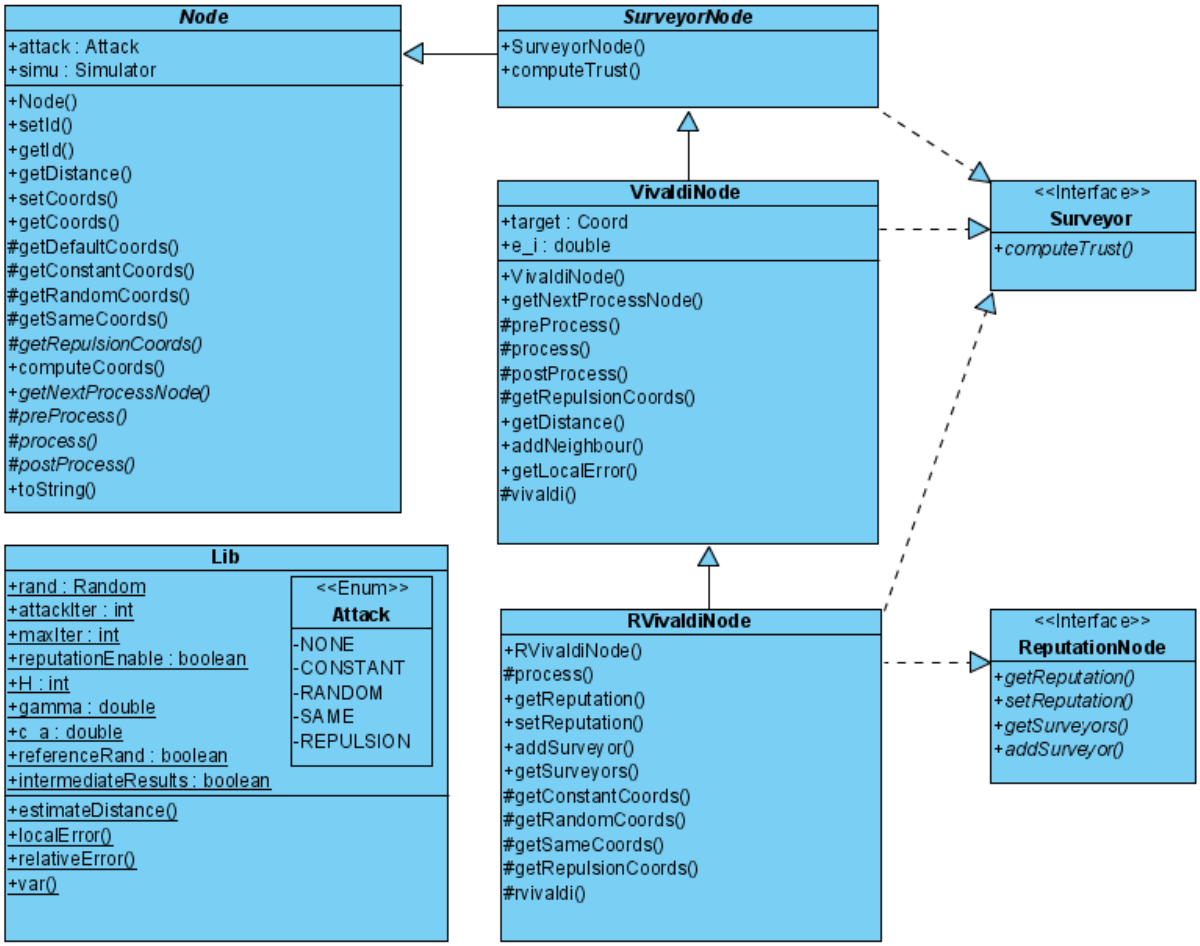


Figure 5.1: Class diagram of the most important classes of the simulator

To define the attack of a particular node, its attribute `attack` (inherited from `Node`) must be set to a value listed in the enumeration `Attack` defined in `Lib`. A value of `NONE` means that the node works correctly. The methods `getDefaultCoords`, `getConstantCoords`, `getRandomCoords`, `getSameCoords` and `getRepulsionCoords` implements the coordinates given by good, constant attack, random attack, same attack and repulsion nodes respectively. To modify how these attacks behave on coordinates, these methods must be overridden by the new implementation. If the attack has influence on the distance, the method `getDistance` has to be overridden. To activate the reputation, the attribute `reputationEnabled` of `Lib` must be set to `true`.

For more details about the simulator, we invite the reader to read the electronic documentation given at `http://ece.fsa.ucl.ac.be/dsaucez/thesis/simulator/doc` or read the source code given in Appendix B or at `http://ece.fsa.ucl.ac.be/dsaucez/thesis/simulator/src`.

Network coordinate systems are concurrent environment by definition. However, we implement the simulator with a single iterative process. The simulator is composed of a main loop that runs 4000 times (twice the number of iterations needed for Vivaldi convergence). At

each iteration, each node is asked to re-compute its coordinates. Listing 5.1 shows the detail of this loop. Lines 3 and 4 indicate that nodes compute their coordinates successively and not concurrently as in a real environment. Moreover, Listing 5.2 shows how a node computes its coordinates when the method `computeCoord` is called on it. When a node computes its new coordinates, it first chooses a neighbour among its list of neighbours. It then computes its coordinates using the RVivaldi algorithm. When the new coordinates are chosen, the node asks the RCA to validate them (line 4).

Listing 5.1: Main loop

```
1 i = 0
2 while(i++ < 4000)
3   for(node : nodes)
4     n.computeCoords()
```

Listing 5.2: Computation of coordinates

```
1 computeCoords(){
2   neighbour = getNextNeighbour()
3   newCoord = rvivaldi(neighbour)
4   rca.validate(this, newCoord)
5 }
```

In our implementation, `getNextNeighbour` selects the next neighbour sequentially among the list of neighbours of the nodes: Call after call, the method returns the next neighbour in the list. When the end of the list is reached, the methods returns to the beginning of the list.

A node calls the method `getCoords` on its neighbour to retrieves the coordinates and calls the method `getDistance` to have the RTT. The attacks are implemented in these methods. Fake coordinates are computed and returned in `getCoords` and the modification of the RTT is done in `getDistance`.

We have presented above that nodes ask the RCA to validate their coordinates, listing 5.3 presents this validation process.

Listing 5.3: Validation of coordinates

```
1  // @param node the node to validate
2  // @param newCoord the new coordinate of the node
3  validate(node, newCoord){
4    stack = new Stack()
5    // retrieve the surveyors
6    surveyors = node.getSurveyors()
7    // compute value of N
8    n = computeN(surveyors)
9    for(surveyor : surveyors){
10     // compute opinion RCA has in surveyor
11     wRCAS = reputationRCA2S(surveyor, node, n)
12     // compute opinion surveyor has in node
13     wSA = reputationS2A(surveyor, node, n)
14     // add the discounting of these opinions in the stack
15     stack.push(wRCAS.discounting(wSA))
16   }
17   // compute the consensus for all the opinion computed above
18   while(stack.size() > 1){
19     stack.push(stack.pop().consensus(stack.pop()))
20   }
21   // transform the opinion in a scalar value
```

```
22    reput = stack.pop().toTrust()
23    // set the reputation to the node
24    node.setReputation(reput)
25    // set the coordinate to the node
26    node.setCoords(newCoord)
27  }
```

The loop in between line 9 and 16 computes $\tilde{\omega}_s^{RCA} \otimes \hat{\omega}_A^s$ for each surveyor s of the node. The resulting opinion for each surveyor is pushed on the opinion stack.

The loop in between line 18 and 20 computes the reputation of the node presented in Eqn. 4.11. This value is obtained by popping the first two opinions ω_1 and ω_1 on the stack, computing $\omega_3 = \omega_1 \oplus \omega_2$ and pushing ω_3 on the stack. When the stack only contains 1 element, this last is the reputation of the node.

The line 22 transforms the reputation in a scalar value compatible with the notion of reputation in RVivaldi. With lines 24 and 26, we can see that the RCA sets the reputation and the coordinates to the node. In a real system, the RCA would have sent a packet containing the reputation and the node would have set its reputation to the received value.

The opinions computed on lines 11 and 13 in Listing 5.3 depends on the computation of the trust presented in Eqn.4.6 and the doubt in Eqn. 4.9. These two values depend on the previous h experiences. The naive solution that computes them using these equations is not optimal. Actually, the calculus time is proportional to the size of the history. Linear time looks not so bad, but in Listing 5.1 we can see that the simulator calls `computeCoord` on every node at each iteration and `computeCoord` call `validate` on the RCA. This methods must compute multiple opinions each time it is called. If we keep the computation of the trust and doubt in linear time, a simulation with reasonable number of nodes would take too long time to complete. The solution we found is to compute the trust and doubt in constant time.

It is possible to rewrite Eqn. 4.6 of trust iteratively as follow:

$$\tau(A,B,t) = \left(\left(\frac{\tau(A,B,t-1)}{a(t-1)\cdot\gamma} - (1-\gamma)^h \cdot \xi(A,B,t-h) \right) \cdot (1-\gamma) + \xi(A,B,t) \right) \cdot a(t) \cdot \gamma. \tag{5.1}$$

However, the iterative version of the trust requires to know the experience value at time $t-h$. Thus, we have to introduce a technique that is still in constant time. Even if the temporal complexity can be constant, it is not possible to have less than a linear space complexity. The idea of the algorithm is to remember the last value of the trust and to keep in memory the last h experiences. At each trust computation with a new experience, the oldest experience is removed from the memory and the new experience is added as the newer one. The data structure that keeps the last h experiences has a $\mathcal{O}(1)$ time complexity for inserting a new experience and the same constant time to access/remove the oldest experience in the structure. The chosen solution is to use a fixed size array with circular indexes. The trust computation also requires to know the age factor associated with the trust. This factor is stored in a specific variable updated when required.

The doubt is a function of the variance (with $N-1$ free degrees) of the last h experiences.

The doubt component of an opinion depends on the variance of the experiences (see Eqn. 4.9). The perfect knowledge of the variance of a sample of h elements is traditionally computed with algorithms in $\prime(h)$ time. To compute the variance of the history in constant time, we use the following approximation:

$$n = \text{the number of remembered experiences} \tag{5.2}$$

$$S_1 = \sum_{i=t-n}^{t} \xi(A,B,i) \tag{5.3}$$

$$S_2 = \sum_{i=t-n}^{t} \xi(A,B,i)^2 \tag{5.4}$$

$$\sigma^2 \approx \frac{S_2}{n} - \left(\frac{S_1}{n}\right)^2 \tag{5.5}$$

This approximation computes the variance of a sample in constant time. In our implementation, we already store the last h experiences and know the size of the set which is the number of experiences stored. To approximate the variance, we add two special variables that store S_1 and S_2. Every time a new experience is added, the value of the new experience is added to S_1 and if their already is h elements in the history, the removed experience is subtracted from S_1. The same procedure is applied to S_2 with the square value of the experiences.

The iterative version of the variance can be written as follow:

$$S_1(t) = \begin{cases} S_1(t-1) + \xi(A,B,t) - \xi(A,B,t-h) & \text{if } n \geq h \\ S_1(t-1) + \xi(A,B,t) & \text{otherwise} \end{cases} \tag{5.6}$$

$$S_2(t) = \begin{cases} S_2(t-1) + \xi(A,B,t)^2 - \xi(A,B,t-h)^2 & \text{if } n \geq h \\ S_2(t-1) + \xi(A,B,t)^2 & \text{otherwise} \end{cases} \tag{5.7}$$

We saw that it is possible to compute the trust and the doubt in constant time. However, the validation also require to know the value of n which is the maximum among the maximum of the different history of experiences used for the validation. We have added a parameter that keeps the maximum value in the history but we have to deplore that it is impossible to keep this variable correct in constant time. However, the algorithm we used is able to compute the maximum of an history in constant time on average. The maximum is computed at each time a new experience is added. If the new experience has higher value than the current maximum, the maximum is set to the value of the new experience. If the new experience is less than the current maximum, two cases are possible. First, if the maximum is higher than the removed value, their is nothing to do. But if the maximum as the same value as the removed element, the history must be seek to find the new maximum of the history. This algorithm is presented in Listing 5.4. If the number of experiences in the history is less than H, the algorithm is limited to lines 6 and 7.

Listing 5.4: Maximum value in the history of experiences

```
1 // @param newExperience the new experience added
2 // @param removedExperience the removed experience
3 // @param currentMaximum the current maximum
4 // @param history the history of experiences
```

```
computeMax(newExperience, removedExperience, currentMaximum, history){
  if(newExperience >= currentMaximum)
    return newExperience
  if(removedExperience == currentMaximum){
    max = newExperience
    for(exp : history){
      if(exp > max)
        max = exp
    }
  }
  return max
}
```

A good reputation system must have dynamic attribution of surveyors and propose a system that make impossible for a malicious node to know the surveyors of a node. However, our simulator does not require this feature because their is no attack on surveyors.

5.3 Performance indicator

As in previous studies of Vivaldi [2, 11, 4, 16], the relative error is our main performance indicator. The relative error gives an information about how the predicted distance is accurate compared to the measured distance. The lower the relative error, the accurate the coordinates.

A way to analyze the accuracy of a coordinate system is to analyze the *cumulative distribution function* (CDF) of the relative error of the victims. An accurate NCS should propose low relative errors. This is why the CDF of the relatives errors is a good indicator of the accuracy because it represents the cumulative portion of nodes for a given relative error. The faster the CDF tends to 1, the accurate the coordinates are. Unfortunately, coordinate systems are not perfect and thus do not have a constant CDF of 1.

Our second performance indicator is inspired by the *relative error ratio* (ratio for short) proposed by Kaafar et al. [4]. The ratio is the average relative error of the system in presence of malicious nodes divided by the average relative error of the system in absence of attacker. We adapt this indicator to observe the impact of the reputation on the accuracy of the system. The new indicator, called *reputation ratio*, is the ratio of the average relative error during an attack when the reputation is enabled with the average relative error during the same attack when the reputation is disabled. The reputation improves the accuracy of the system if its reputation ratio is below 1. On the contrary, a reputation ratio larger than 1 shows a diminution of accuracy compared to the system in absence of protection.

The objective of the reputation model is not to improve the accuracy of the network coordinate systems, but to limit the impact of attacks. So that, we always compare the relative errors with and without reputation.

Before analysing the results of the simulations, we must define what is the best combination of the reputation parameters. A good choice is a choice that always presents lower relative errors than when the reputation is disabled. The best choice is the choice that presents the lower relative errors on average. To study the impact of the choice of parameters, we divide the experiments in *sets*. Each set differs from others by the choice of the parameters. Regarding

the evolution of the CDF of relative errors for all the sets shows that some choices outperform some other. Sec. 5.4 presents the results.

5.4 Results

In this section, we study the results obtained for the 1200 simulations. First, we observe how the system behaves for the different combinations of parameters. Based on the observations, we choose the best value for the parameters γ, h and c_a. Next, we analyze the impact of the reputation on the accuracy of the coordinates with these parameters.

5.4.1 Vivaldi

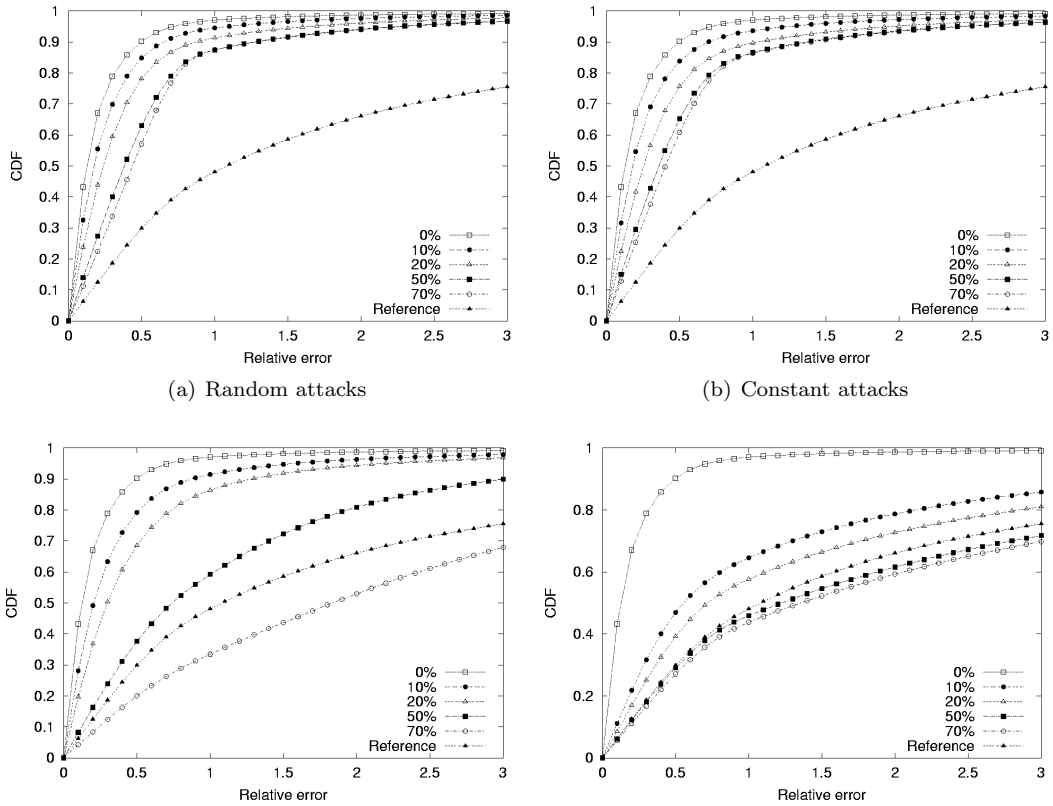
In this section, we present the results of the four attacks on Vivaldi using our simulator.

Fig. 5.4.1 depicts the CDFs of relatives errors without the reputation mechanism for different number of attackers. The curve labeled *reference* is the CDFs of the relatives errors when the coordinates of nodes are randomly chosen (i.e., Vivaldi is not executed). We plot results for the random attack (Fig. 5.2(a)), the constant attack (Fig. 5.2(b)), the same attack (Fig. 5.2(c)) and the repulse attack (Fig. 5.2(d)).

With these CDFs, we see that RVivaldi is more sensitive to same attacks and repulse attack than to random attacks or constant attacks. We also observe that Vivaldi remains accurate even with more than 50% of malicious nodes leading the last two attacks. Remember that a value of 0 of the relative error means the predicted distance and the measured distance are equals. A value of a for the relative error means that the predicted distance differ by a times the measured distance (e.g., a relative error of 1 means that the predicted distance is the double of the measured distance). In random and constant attacks, more than 85% of nodes have a relative error lower or equal to 1. In other words, the predicted distance in more than the double of the measured distance for only 15% of the nodes. However, in presence of 70% of malicious nodes, only a portion of 10% of the correct nodes are really disturbed and present bad predicted distances.

For same attacks, the results are worse. We observe that Vivaldi coordinates are worst than randomly chosen coordinates when 70% of the nodes are malicious (i.e., the curve for 70% of malicious nodes is below the reference curve). We can also see that 45% of nodes are strongly affected in presence of 50% of malicious nodes and an important portion of 12% of nodes are affected by only 20% of malicious nodes. These observations where predictible because every time a good node computes its coordinates with a same coordinates attacker, the good node is forced to move from its current position.

Worst, for more than 30% of good nodes in a repulse attack scenario, the predicted distance is more than the double of the measured distance in presence of only 10% of malicious nodes. Compared to the less than 5% of nodes in absence of attackers, the same attack seems to be particulary efficient on Vivaldi. In addition, in presence of 50% or more attackers, the systems behaves worst than randomly chosen coordinates. In other words, even in presence of few attackers, less than 65% of the good nodes have a reasonable predicted distance inferior to the double of the measured distance.



(a) Random attacks

(b) Constant attacks

(c) Same attacks

(d) Repulse attacks

Figure 5.2: CDF of relative errors in Vivaldi when the reputation is disabled

Another difference between constant or random attacks and repulse or same attacks is that for the first two, the CDFs of relative errors rapidly converge to the same value as Vivaldi in absence of attackers. This observation can be explained by the fact that the impact of a random or a constant attack is less important than the inherent inaccuracies of Vivaldi: Some nodes may present strongly inaccurate coordinates. On the contrary, for the two last attacks (i.e., same and repulse attacks), the relative error slowly converges to the same value as Vivaldi in absence of attackers. Indeed, these attacks cause more important errors that inherent errors due to Vivaldi and the quality of coordinates is more affected by the attacks that by the approximations done by Vivaldi.

For all these attacks, the most frequent the malicious nodes, the more the relative error. This property of Vivaldi is important since it shows that a single node is not able to have an important impact on the accuracy of the whole system. Kaafar et al. show that if the number of node in Vivaldi increases, the sensibility to attacks decreases (e.g., a system of 100 nodes is more sensible to an attack performed by 10 nodes than a system with 10000 nodes with 1000 attackers) [16].

5.4.2 RVivaldi: Choice of γ, h and c_a

The reputation model is based on three parameters: γ, h and c_a. These parameters give 60 possible combinations with the domains shown in Table 5.1. In this part of the work, we discuss the choice of these parameters.

For each attack, we adopt the same methodology. We plot the cumulative number of nodes for a relative error of 0.5, 1 and 1.5 for the 60 possible combinations of parameters. The combinations are ordered on the X-axis and the Y-axis represents the value of the CDF for the relatives errors of a given set (Relative errors summary). For readability, we group the different values of CDFs for a given relative error by value of c_a. The value of the age bonus coefficient of the group of sets is written on the X-axis at the top of the figure. Every group of sets with a given c_a is divided in groups of set with the same value of γ. The main tics in the X-axis at the bottom of the figure represents a group of set with a given γ. Each group is then divided by value of h. The secondary tics in the X-axis at the bottom of the figure represents a value of h. The value of the parameters are ordered is ascending order from left to right modulo the metric.

For example, from left to right, the points in a group of a given c_a corresponds to the choice of parameters $\langle\gamma = 0.01, h = 10\rangle$, $\langle\gamma = 0.01, h = 20\rangle$, $\langle\gamma = 0.01, h = 40\rangle$, $\langle\gamma = 0.1, h = 10\rangle$, $\langle\gamma = 0.1, h = 20\rangle$, $\langle\gamma = 0.1, h = 40\rangle$, $\langle\gamma = 0.2, h = 10\rangle$, $\langle\gamma = 0.2, h = 20\rangle$, $\langle\gamma = 0.2, h = 40\rangle$, $\langle\gamma = 0.5, h = 10\rangle$, $\langle\gamma = 0.5, h = 20\rangle$ and $\langle\gamma = 0.5, h = 40\rangle$ respectively.

The CDFs value for the three relative errors in absence of reputation are also plotted to see if the reputation improves the accuracy or not. A relative error of 0.5 is considered as low and a relative error of 1 or more is considered high as it means that the predicted distance is known with an error of more than 100%. The best choice of the parameters is those which systematically presents higher value for the CDF for every relative error.

Before comparing the sets for the four attacks, Fig. 5.3 depicts the results in absence of malicious nodes. We can observe that RVivaldi does not outperform Vivaldi. Worse, with $\gamma = 0.01$ RVivaldi computes really bad coordinates compared to Vivaldi (up to 10% worst for relative error of 0.5). For small relatives errors, the diminution or improvement of accuracy due to the reputation is more important than for high relative error. The explication is the same as for the impact of attacks on Vivaldi. Indeed, the high relative errors are not frequent and are the results of the innacuracy inherent to the prediction model.

Comparing the 5 groups of age bonus coefficient, we could observe that c_a does not affect the quality of the distance estimator. This result was predictable since the age bonus coefficient was introduced to protect the system against nodes with multiple identities and this attack is not modeled by the simulator.

Random attack: The next 4 figures present the evolution of the CDFs in presence of malicious nodes performing random attacks. In Fig. 5.4 the system is attacked by 10% of malicious nodes. Fig. 5.5 considers 20% of attackers. Next, we show the evolution of the CDFs for 50% of malicious nodes in Fig. 5.6. Finally, Fig. 5.7 illustrates the impact of 70% of malicious nodes.

Fortunately, when attackers are in the system, the reputation may improve the accuracy. In Fig. 5.4, we observe the modification of CDF in presence of 10% of malicious nodes. Once again, c_a does not change the accuracy of the coordinates. For $\gamma = 0.01$ and $h \leq 20$ the results are worst than with Vivaldi. The choice of $\gamma = 0.1$ offers the best results independently of the choice of h. On the contrary, for higher values of γ (i.e., $\gamma \in \{0.1, 0.2, 0.5\}$), RVivaldi

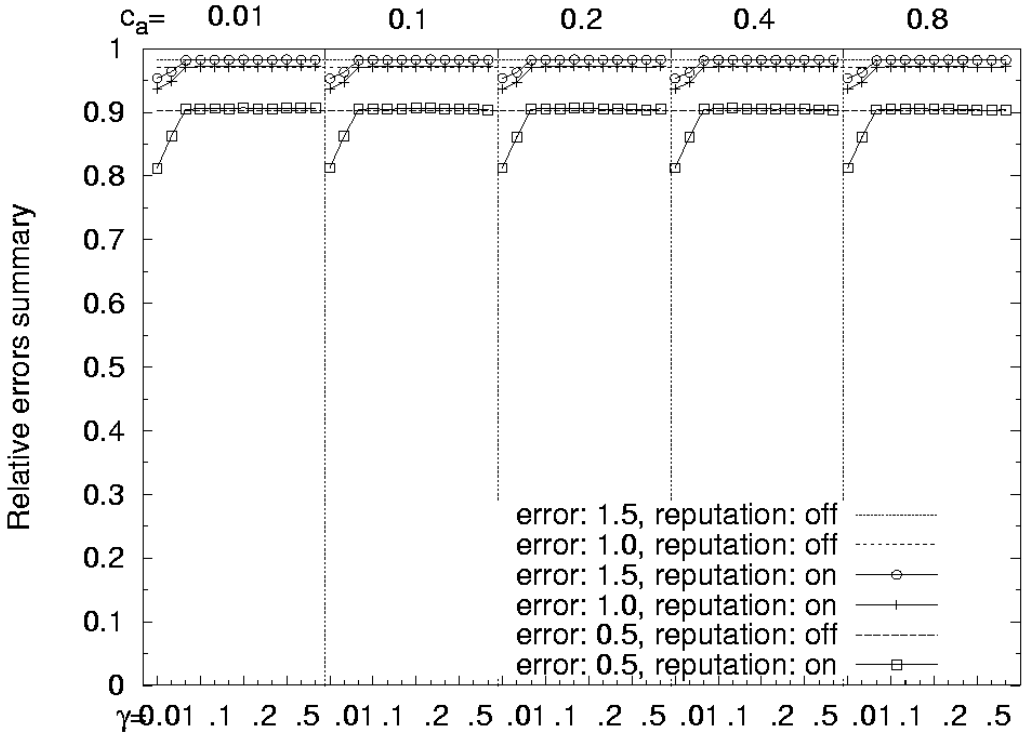


Figure 5.3: Evolution of the CDF with the parameters in absence of malicious node

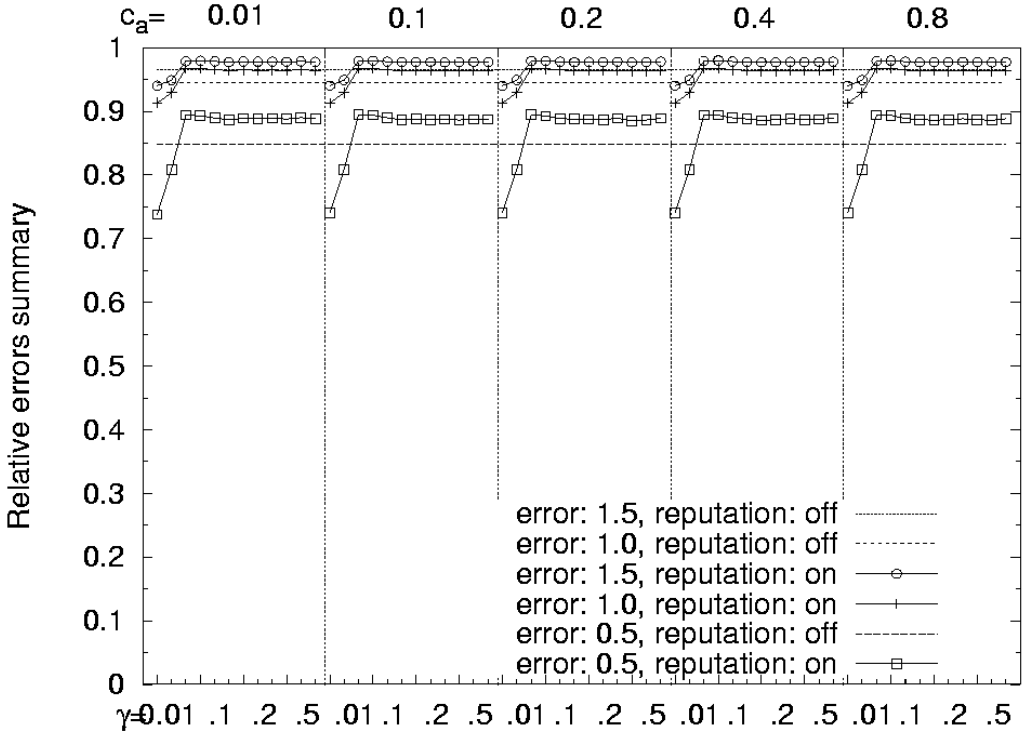


Figure 5.4: Evolution of the CDF with the parameters for 10% of random attackers

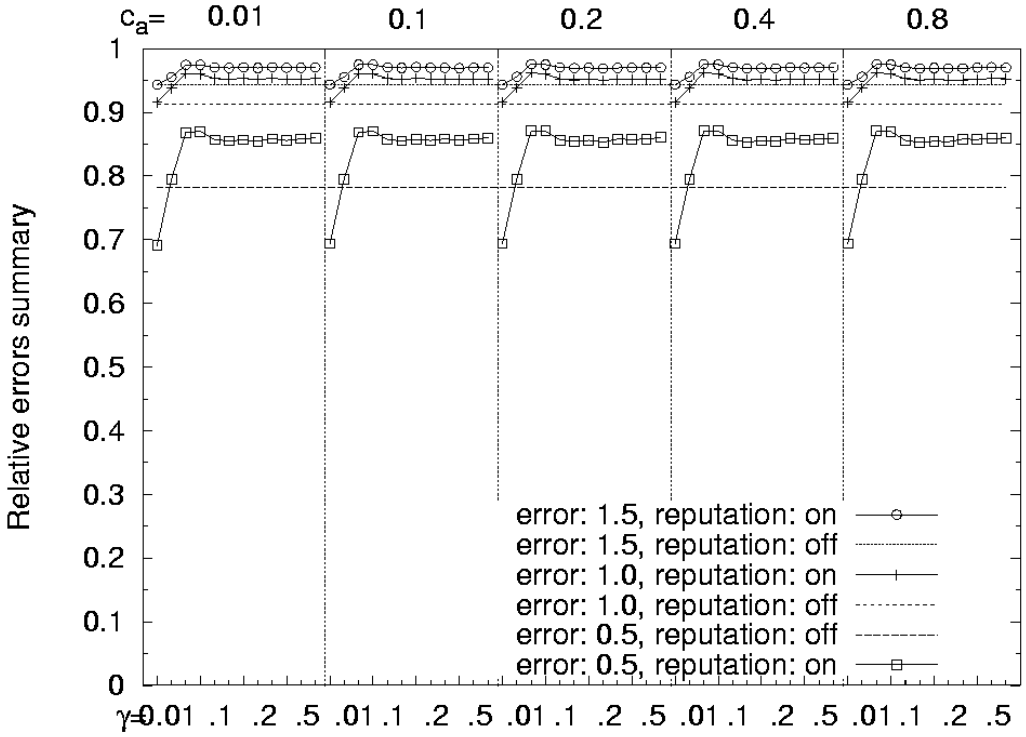


Figure 5.5: Evolution of the CDF with the parameters for 20% of random attackers

outperforms the CDF of Vivaldi by up to 5% for a relative error of 0.5. The best result is always obtained for $\gamma = 0.1$ and $h = 10$ whatever the relative error. It is necessary however to moderate this result. Indeed, when $\gamma \geq 0.1$ the difference between the best result and the worst is less than 1%. In addition, the overall difference for a relative error of 1 is around 5%. 8

Interestingly, in presence of 20% of malicious nodes (Fig. 5.5), RVivaldi always equals or outperforms Vivaldi for relative error higher than 1. For the third time, we can observe that c_a does not change the accuracy of RVivaldi. We observe the same behavior than with 10% of attackers. The worst case is obtained for $\gamma = 0.01$ and $h = 10$ and the most accurate coordinates are computed when $\gamma = 0.1$ and $h = 10$. The maximum difference of accuracy introduced by the choice of γ and h is around 2%. We can also observe that coordinates for small relative errors are better than with 10%.

In presence of 50% or 70% of attackers (Fig. 5.6 and Fig. 5.7, respectively), c_a has no influence on the accuracy. For a relative error of 0.5 RVivaldi behaves like Vivaldi for 10% or 20%. However, the situation changes for relative errors of 1.0 or 1.5. Indeed, the best accuracy is obtained for $\gamma = 0.01$ and $h = 10$. The worst results are obtained for $\gamma = 0.5$ whatever h. We must however moderate this conclusion because the difference between the best case and the worst case is less than 3%. On the contrary, for a relative error of 0.5, the bad choice of $\gamma = 0.01$ is 15% worst that the best choice for 50% of malicious nodes. For 70% of malicious nodes and the same relative error, the difference is of 18%. For a low relative error of 0.5, the best choice is $\gamma = 0.2$ and $h = 40$.

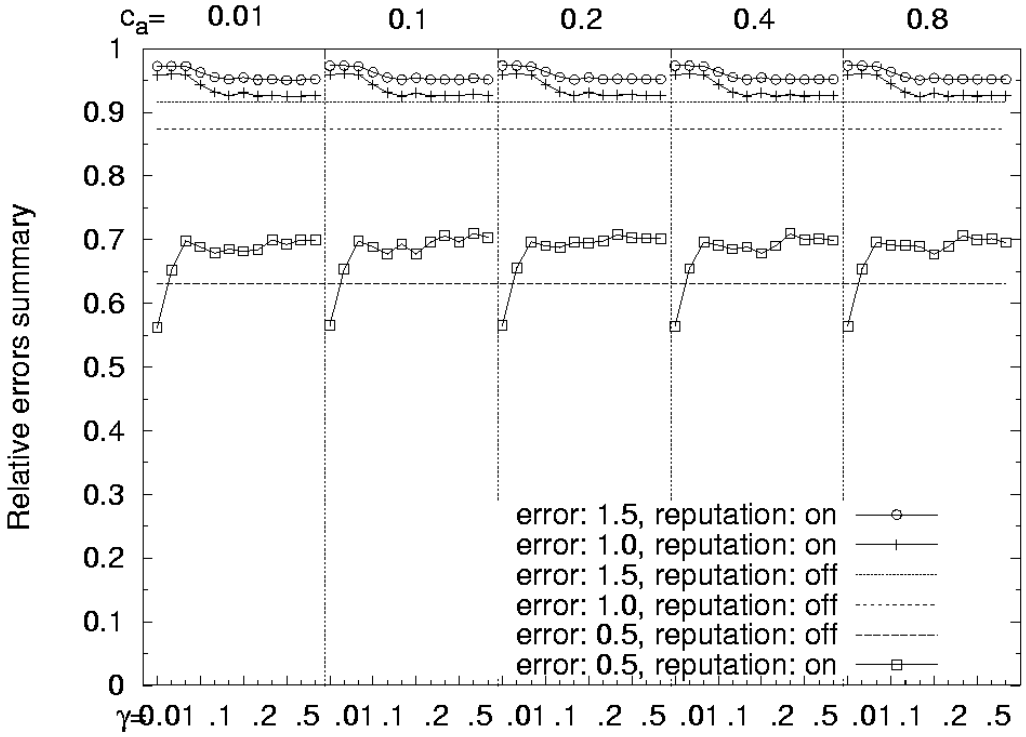


Figure 5.6: Evolution of the CDF with the parameters for 50% of random attackers

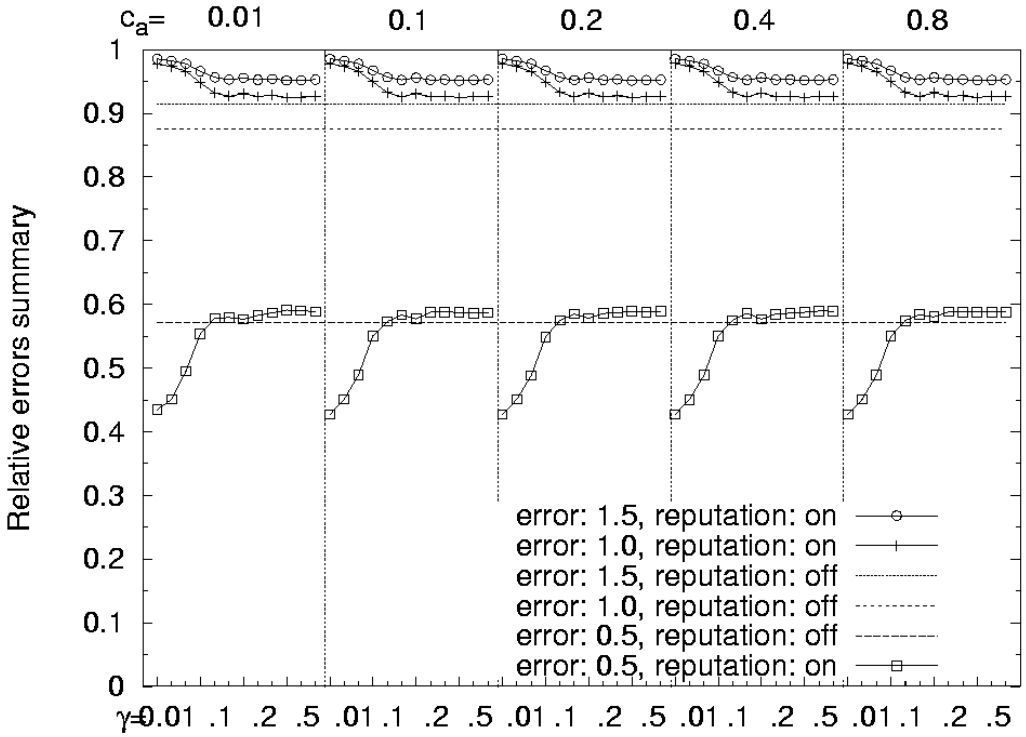


Figure 5.7: Evolution of the CDF with the parameters for 70% of random attackers

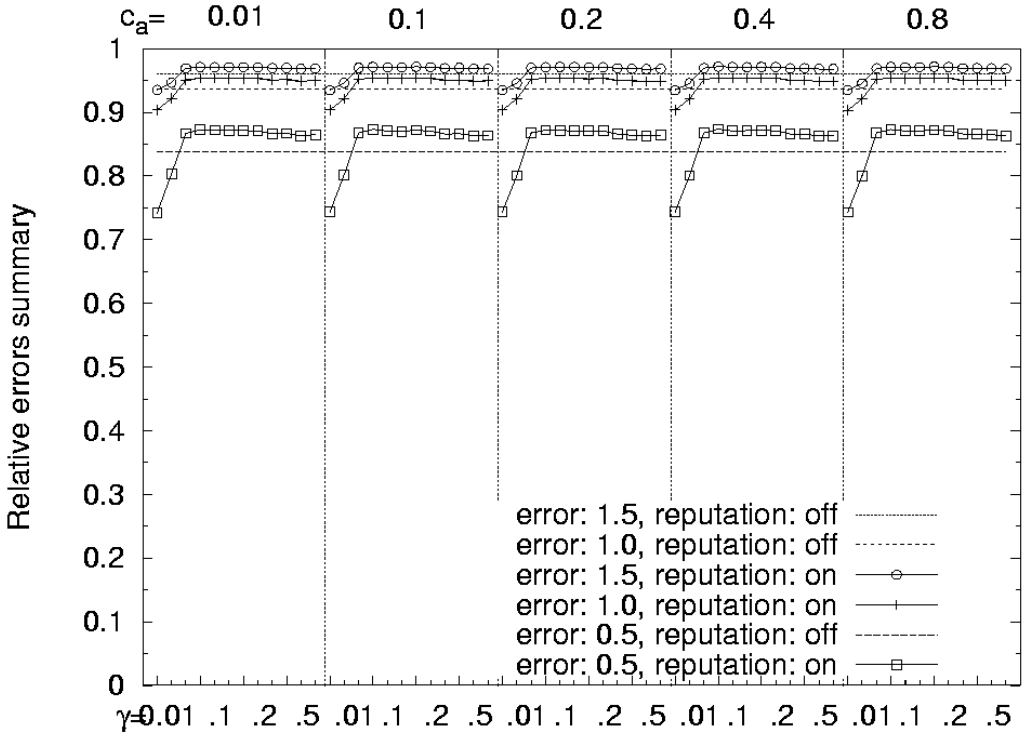


Figure 5.8: Evolution of the CDF with the parameters for 10% of constant attackers

We have seen that c_a has no influence on the accuracy of coordinates. We propose to use a small value of $c_a = 0.01$ or lower to limit the possible impact of multiple identities. In presence of malicious nodes, RVivaldi can outperforms Vivaldi. However, the quality of RVivaldi depends on the choice of γ and h. Globally, the best choice is $\gamma = 0.1$ and $h = 10$, but this choice is not always the best.

If the number of attackers is known, the choice of γ and h can be improved. For instance, if many malicious nodes are present (i.e., 50% of more), a choice of $\gamma = 0.01$ and $h = 40$ is desirable. But with 70% of malicious nodes, the choice depends on what to optimize. If we want to maximize the number of nodes with a low relative error, the best choice is $\gamma = 0.2$ and $h = 40$. On the contrary, if we want to minimize the number of nodes with a high relative error, the best choice is $\gamma = 0.01$ and $h = 10$.

In the rest of this chapter, we always consider the choice of parameters that gives the better results on average. **For random attacks, this choice is $\gamma = 0.1$, $h = 10$ and $c_a = 0.01$.** RVivaldi always outperforms Vivaldi with this choice of parameters.

Constant attack: The next 4 figures present the evolution of the CDFs in presence of malicious nodes performing constant attacks. In Fig. 5.8 the system is attacked by 10% of malicious nodes. Fig. 5.9 considers 20% of attackers. Next, we show the evolution of the CDFs for 50% of malicious nodes in Fig. 5.10. Finally, Fig. 5.11 illustrates the impact of 70% of malicious nodes.

The shape of the curve for 10% of constant attacks (Fig. 5.8) is similar to the same curve for random attacks. The age bonus coefficient has no impact on the accuracy of coordinates.

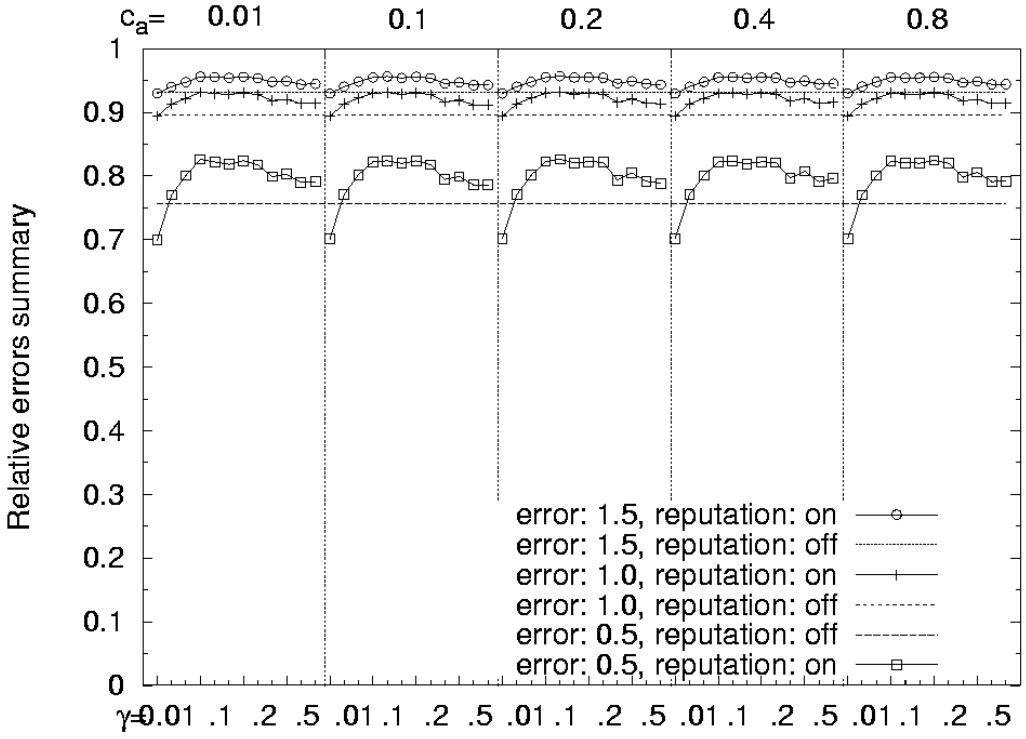


Figure 5.9: Evolution of the CDF with the parameters for 20% of constant attackers

The better coordinates are obtained for $\gamma = 0.1$ and $h = 10$ and the worst case is $\gamma = 0.01$ and $h = 10$. The difference of CDFs is 5% for a relative error of 1 or 1.5 and about 12% for a relative error of 0.5. As in random attacks, a bad choice of $\gamma = 0.01$ and $h \leq 20$ causes worst results than Vivaldi.

In presence of 20% of attackers (Fig. 5.9), the curves are similar but RVivaldi performs worse than Vivaldi only for a small relative error of 0.5 with $\gamma = 0.01$ and $h = 10$. Once again, the lowest portion of nodes with high relative error is obtained for $\gamma = 0.1$ and $h = 10$. The difference between the worst case and the best case for a relative error of 1 or 1.5 is about 3%. For a small relative error, the difference is up to 13%. As expected, c_a does not influence the quality of RVivaldi.

When considering 50% of attackers (Fig.5.10), RVivaldi always outperforms Vivaldi and the best results are obtained for $\gamma = 0.01$ and $h = 10$ for relative errors higher than 1 and with $\gamma = 0.1$ and $h = 10$ for a small relative error of 0.5. The results must however be moderated because the difference of accuracy between the two extreme cases for a relative error of 1 or 1.5 is less than 3%. For a relative error of 0.5 the difference is less than 10%. c_a has no impact on the performance of RVivaldi.

The observations with 50% of misbehaving nodes are the same in presence of 70% of bad nodes (Fig.5.11). The best results are obtained for $\gamma = 0.01$ and $h = 10$ for high relative errors and for $\gamma = 0.1$ and $h = 10$ for low relative errors. Nevertheless, the impact of the value of γ and h is more limited for 70% than for 50% of malicious nodes for low relative error: The extremes present a difference of 7%. Again, the age bonus coefficient has no impact on the

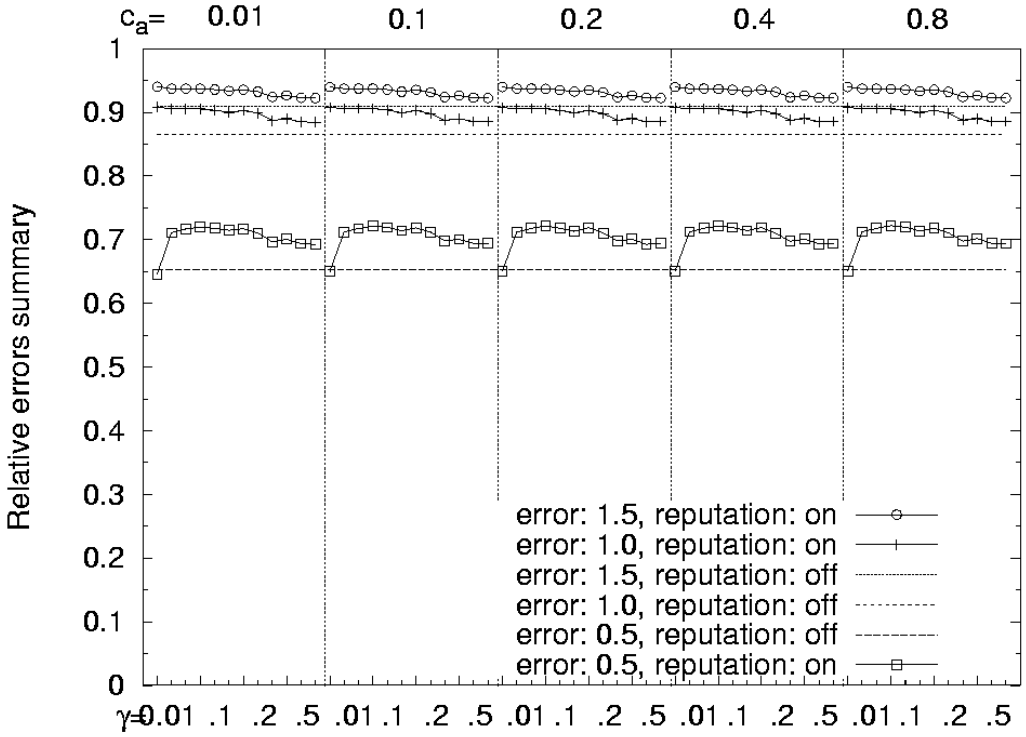


Figure 5.10: Evolution of the CDF with the parameters for 50% of constant attackers

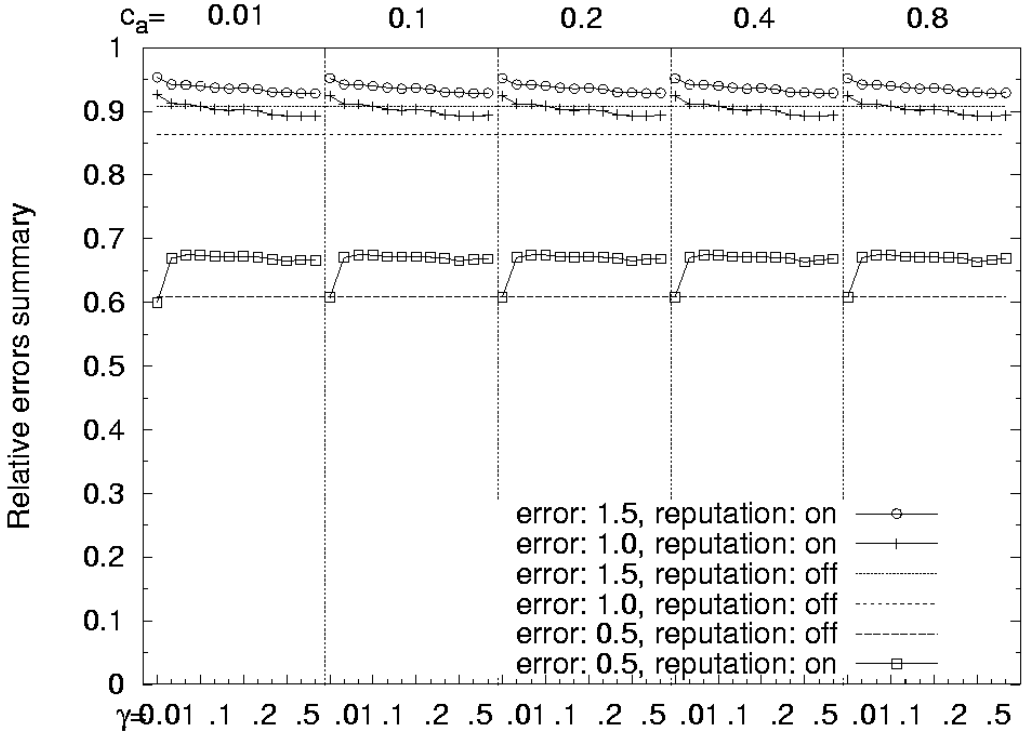


Figure 5.11: Evolution of the CDF with the parameters for 70% of constant attackers

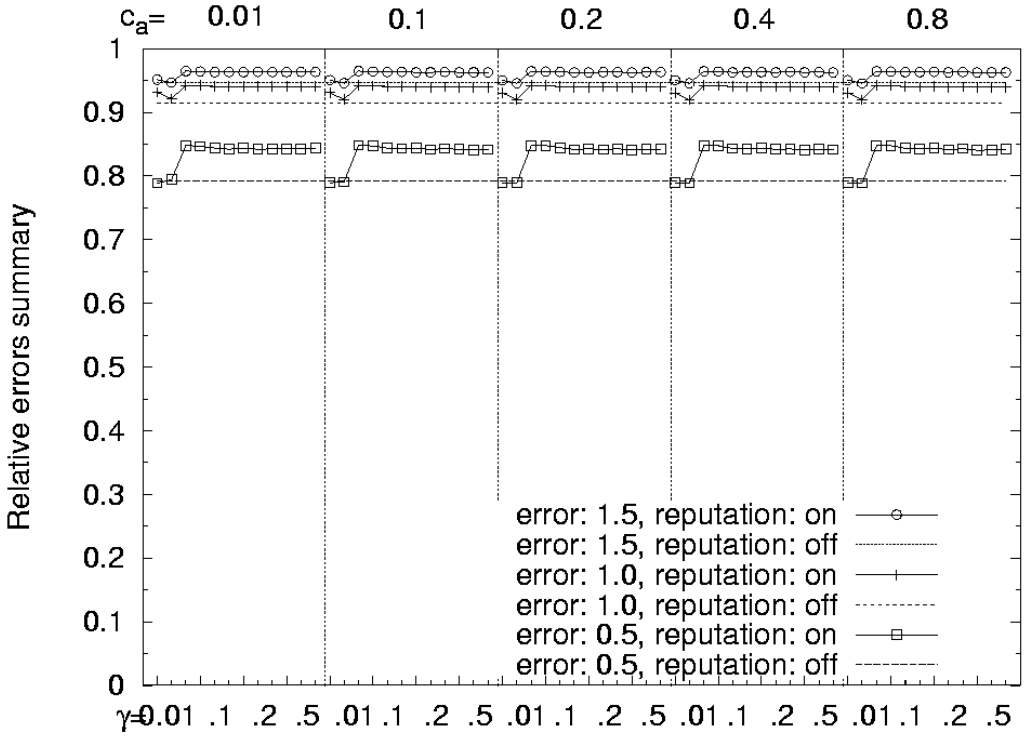


Figure 5.12: Evolution of the CDF with the parameters for 10% of same attackers

coordinates. When considering 50% or more attackers, and if the number of nodes with high relative error has to be minimized, $\gamma = 0.01$ and $h = 10$ is the best choice.

On average, **RVivaldi gives the best results for constant attacks with** $\gamma = 0.1$ **and** $h = 10$ **and is independent of** c_a that we propose to fix at 0.01. With theses parameters, RVivaldi always outperforms Vivaldi.

Same attack: The next 4 figures present the evolution of the CDFs in presence of malicious nodes performing same attacks. In Fig. 5.12 the system is attacked by 10% of malicious nodes. Fig. 5.13 considers 20% of attackers. Next, we show the evolution of the CDFs for 50% of malicious nodes in Fig. 5.14. Finally, Fig. 5.15 illustrates the impact of 70% of malicious nodes.

In presence of 10% of malicous nodes performing a same attack (Fig. 5.12), RVivaldi always outperforms Vivaldi. The best results are obtained for $\gamma = 0.01$ and $h = 40$ or for $\gamma = 0.1$ and $h = 10$. The worst case is when $\gamma = 0.01$ and $h = 20$. For a high relative error of 1 or 1.5, the difference between the best and the worst choice is about 2% and for a low relative error of 0.5 the difference is about 6%. It is interesting to notice that the worst cases of RVivaldi give the same CDF as Vivaldi. As for previous observations, the age bonus coefficient does not influence RVivaldi in our simulations.

In presence of 20% of attackers (Fig. 5.13) and a relative error of 1 or 1.5, the best results are obtained for $\gamma = 0.01$ and $h = 10$ and the worst case is reached at $\gamma = 0.01$ and $h = 20$. On the other hand, for a relative error of 0.5, the worst case is still obtained for $\gamma = 0.01$ and $h = 10$ but the best results are obtained for $\gamma = 0.1$ and $h = 10$. However, the choice of $\gamma = 0.1$

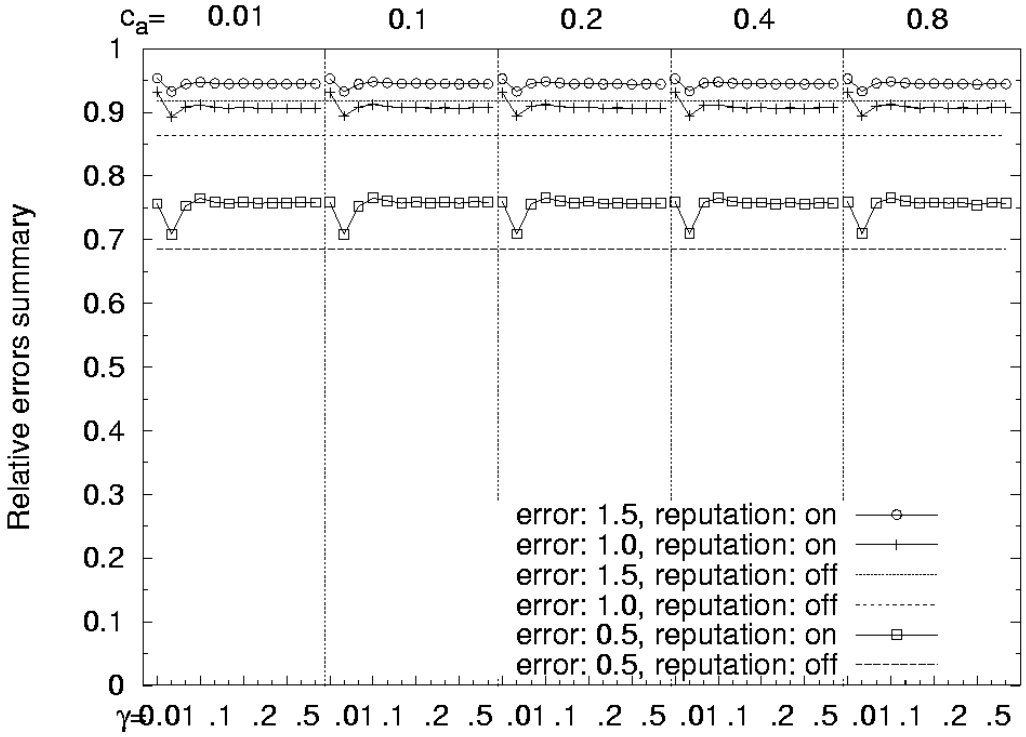


Figure 5.13: Evolution of the CDF with the parameters for 20% of same attackers

and $h = 10$ gives the second best results after the choice of $\gamma = 0.01$ and $h = 10$ for a relative error of 1 or 1.5. In addition, the difference of accuracy between these two results is less than 1%. RVivaldi always outperforms Vivaldi, whatever the choice of γ and h. As expected, the choice of c_a has no real importance in this simulation.

In presence of 50% of attackers (Fig. 5.14), the shape of the curves changes. Indeed, when the results are approximatively the same whatever the choice of γ and h in the previous percentages of attackers, a good choice of γ and h may improve the performance of the system by 30% or even more. Fortunately, the worst case is still better than Vivaldi. The best choice is $\gamma = 0.01$ and $h = 10$. On the contrary, a choice of $\gamma = 0.5$ and $h = 40$ leads to the worst results which outperforms Vivaldi by less than 1%. It is interesting to observe that the worst case for a given γ is always obtained for the higher size of the history ($h = 40$). c_a does not influence the relative errors.

The observation we made for 50% of attackers can be reiterate for 70% of attackers (Fig. 5.15). The best choice is still for $\gamma = 0.01$ and $h = 10$ and the worst case is $\gamma = 0.5$ and $h = 40$. In addition, whatever the value of γ, the worst case is always obtained for $h = 40$. A good choice of γ and h can improve the performance of the system by nearly 50% for high relative errors of 1 or more. The improvement for a low relative error of 0.5 can reach 40% which is not so bad. With 70% of attackers, RVivaldi still outperforms Vivaldi.

In contrast to the random and the constant attacks, it is hard to decide the best choice of parameters for a same attack. In presence of at least 20% of attackers, the choice of $\gamma = 0.01$ and $h = 10$ looks to be the better. This can be explained by the fact that a low weighting constant and a small history gives only few importance to important attacks (only 1% of the

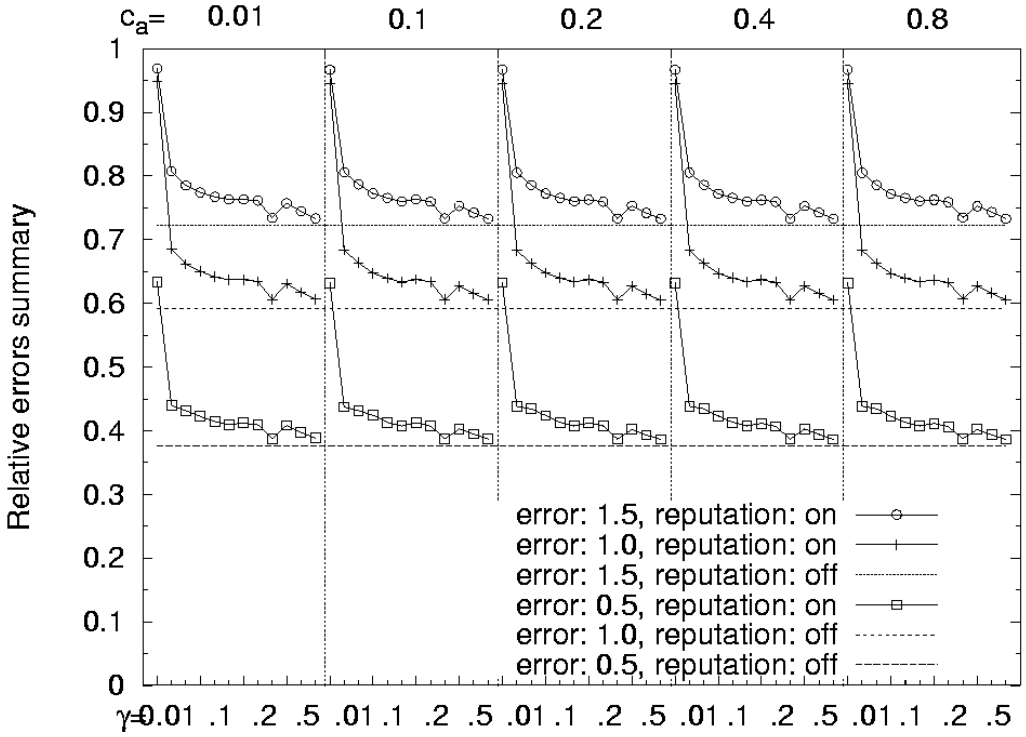


Figure 5.14: Evolution of the CDF with the parameters for 50% of same attackers

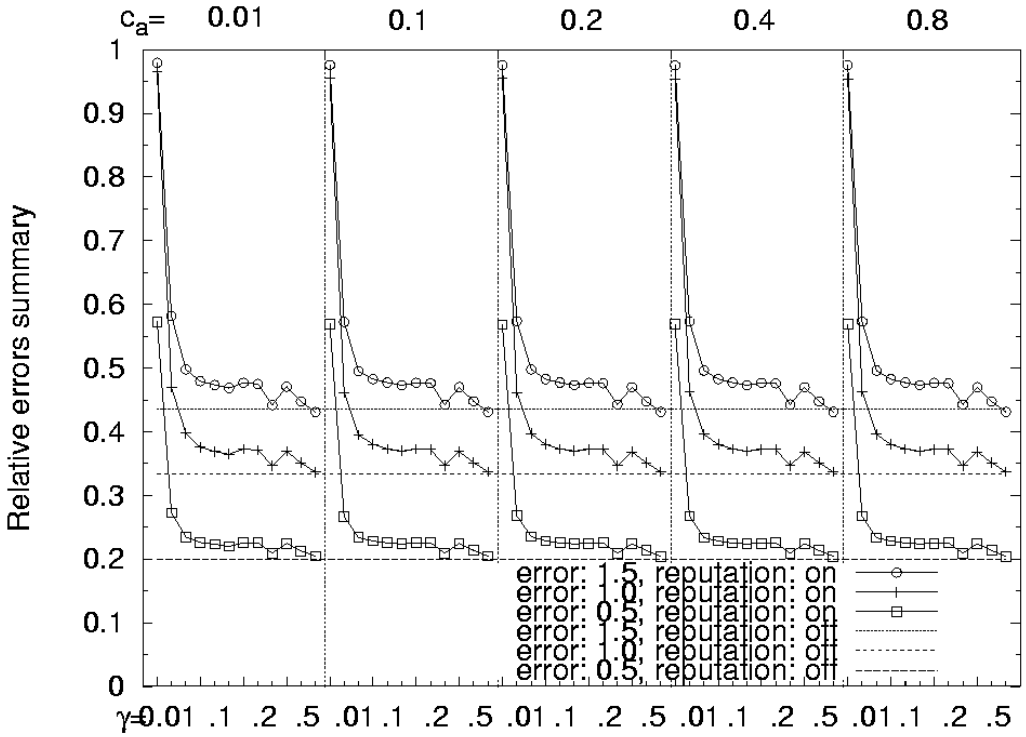


Figure 5.15: Evolution of the CDF with the parameters for 70% of same attackers

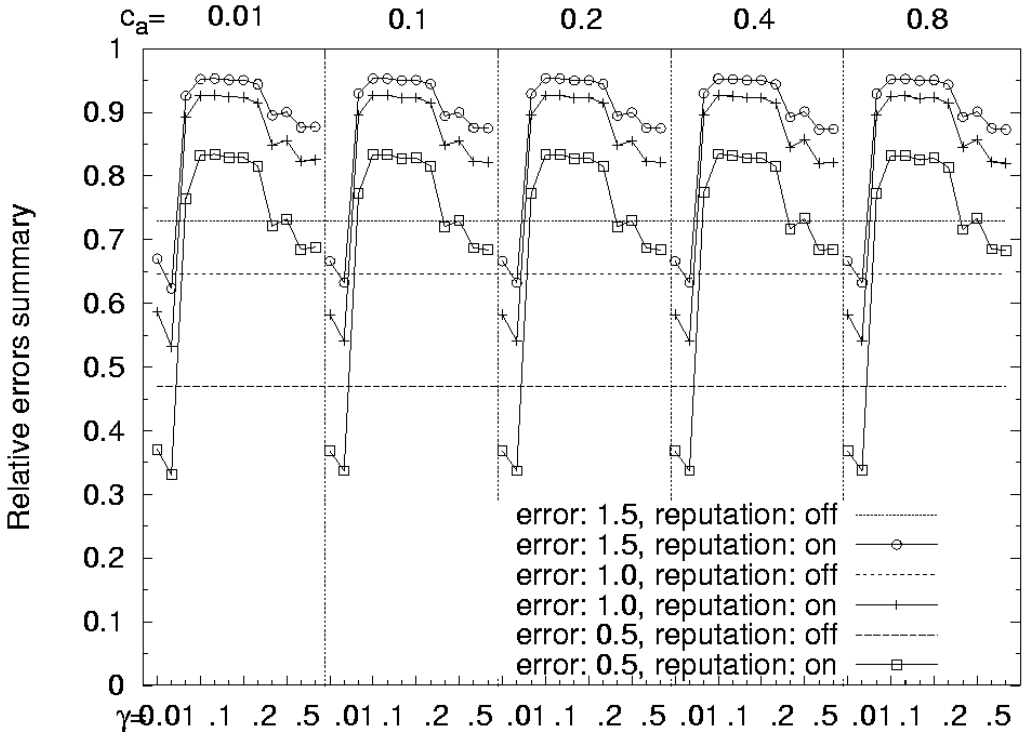


Figure 5.16: Evolution of the CDF with the parameters for 10% of repulse attackers

most recent measurements is used). The problem with this choice is that it give the worst results in presence of only few attackers (less than 10%). Fortunately, RVivaldi always outperforms Vivaldi in such attacks, whatever the combination of parameters. If the number of malicious nodes is unknown, we thus recommend to set $\gamma = 0.01$ and $h = 10$. However, if it is known that the attackers are rare we recommend to set γ to 0.1 and h to 10.

RVivaldi can offer an important improvement in presence of numerous attackers performing same attacks. In presence of 70% of attackers RVivaldi can even be more accurate than Vivaldi for 50% of malicious nodes. Nevertheless, RVivaldi is strongly dependent on the parameters γ and h. In same attacks, the lower the size of the history, the better the accuracy.

By default, **we propose to choose $\gamma = 0.1$, $h = 10$ and $c_a = 0.01$ in presence of same attacks**. This choice is comforted by the fact that it gives the best results when the number of attackers is limited and is always among the four best results when the number of malicious nodes increases. On the contrary, the choice of $\gamma = 0.01$ and $h = 10$ is the best for an important number of attackers but is the worst when the number of attackers is limited. Fortunately, RVivaldi always outperforms Vivaldi, whatever the choice of γ and h.

Repulse attack: The next 4 figures present the evolution of the CDFs in presence of malicious nodes performing repulse attacks. In Fig. 5.16 the system is attacked by 10% of malicious nodes. Fig. 5.17 considers 20% of attackers. Next, we show the evolution of the CDFs for 50% of malicious nodes in Fig. 5.18. Finally, Fig. 5.19 illustrates the impact of 70% of malicious nodes.

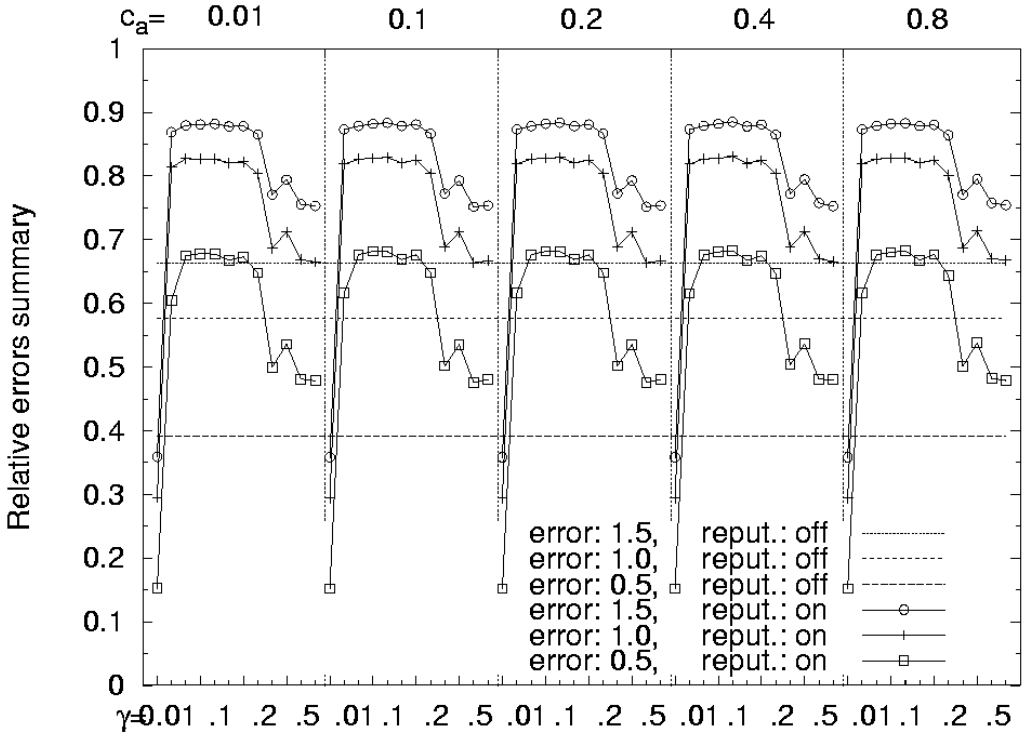


Figure 5.17: Evolution of the CDF with the parameters for 20% of repulse attackers

In presence of 10% of attackers performing a repulse attack (Fig. 5.16), the reputation model behaves in the same manner for any value of the relative error. We observe that the best coordinates are obtained for a choice of $\gamma = 0.1$ and $h = 10$. On the contrary, the worst case is always obtained for the choice of $\gamma = 0.01$ and $h = 20$. RVivaldi outperforms Vivaldi for $\gamma \geq 0.1$ whatever the size of the history. However, the difference between the best and worst choices that outperform Vivaldi is above 10%. For $\gamma = 0.1$, the choice of h has no influence on the accuracy. As for the previous observations, the choice of c_a does not change the accuracy of the coordinates. For low relative errors, the improvement due to the reputation is more important (more than 35%) than for high relative errors (less than 25%). The worst choice of γ and h gives relative error 10% worst than Vivaldi.

Fig. 5.17 depicts the relative errors for 20% of malicious nodes. The observations for 20% of attackers are the same than for 10%. We should notice however that while the worst case was only 10% worst than Vivaldi in the previous observation, the difference reaches 25% for 20% of nodes.

The situation is different for 50% of attackers (Fig. 5.18). Indeed, RVivaldi always outperforms Vivaldi for this number of attackers. Nevertheless, a difference of 10 to 20 percent is observed for the CDFs of relative errors, depending on the parameters choice. For an important relative error of 1 or more, the best choice is $\gamma = 0.01$ and $h = 10$ and the improvement is about 20% compared to Vivaldi. On the contrary, the best choice for a small relative error of 0.5 is $\gamma = 0.1$ and $h = 10$ and the difference with Vivaldi is 13%. This parameters choice corresponds to the third best choice for high relative errors. As expected, the age factor has no impact on the accuracy.

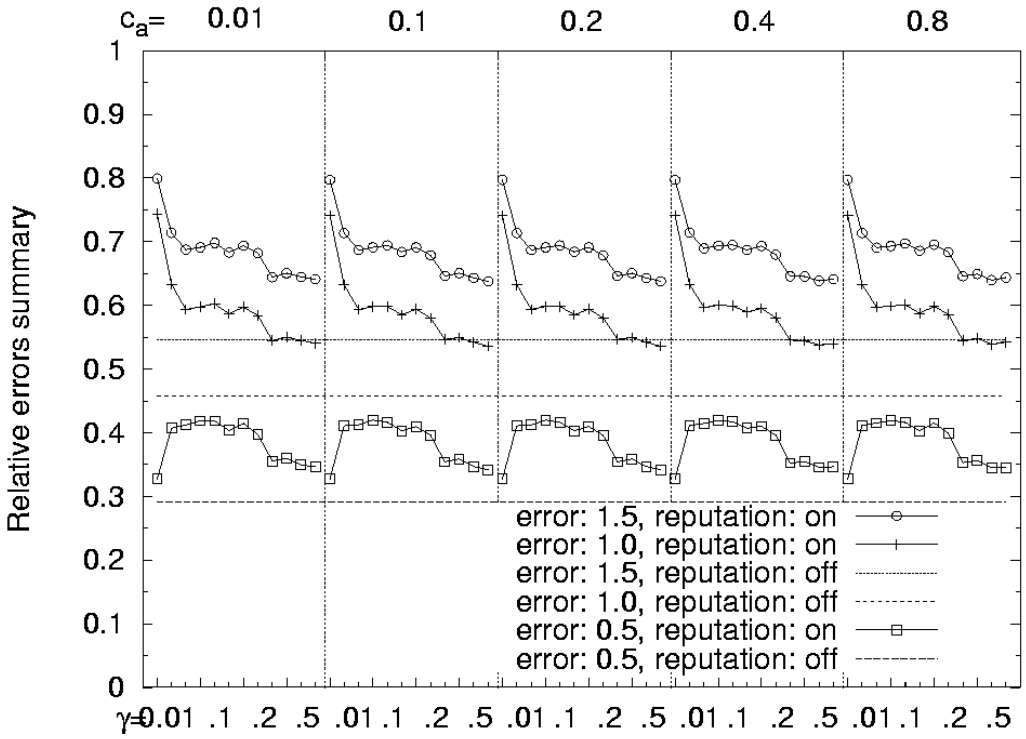


Figure 5.18: Evolution of the CDF with the parameters for 50% of repulse attackers

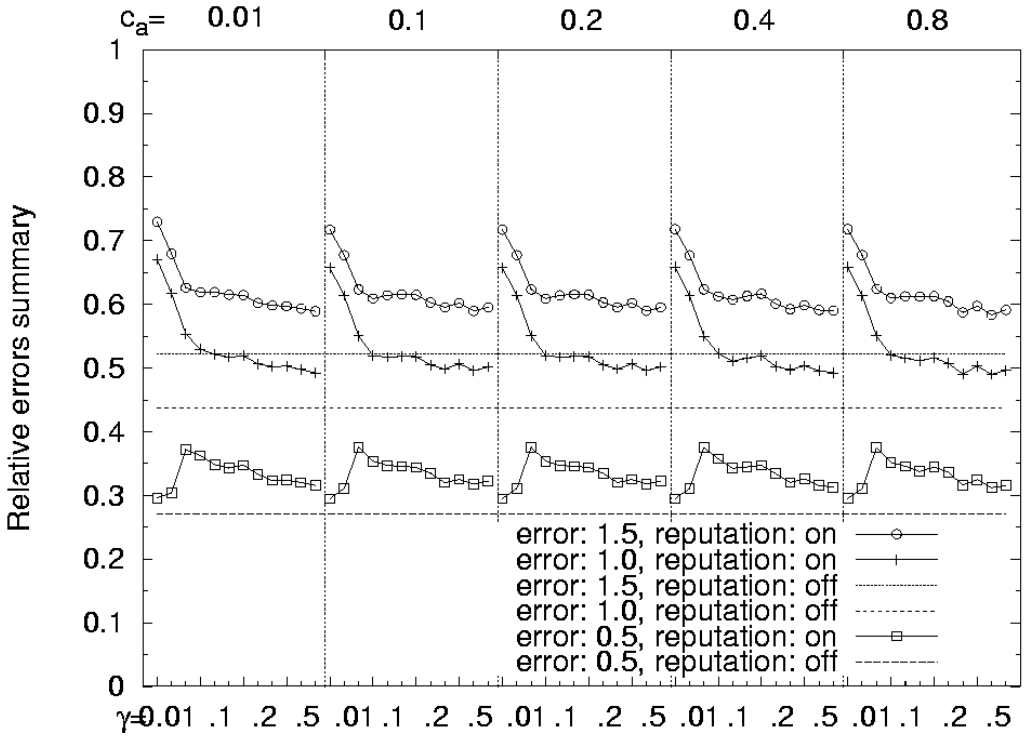


Figure 5.19: Evolution of the CDF with the parameters for 70% of repulse attackers

Fig. 5.19, the best choice of parameters for a relative error of 1 or more is $\gamma = 0.01$ and $h = 10$. In such a case, RVivaldi improves CDFs of relative error by 20% compared to Vivaldi. For a small relative error of 0.5, the best choice of parameters is $\gamma = 0.01$ and $h = 40$ and the improvement compared to Vivaldi is 10%. This choice of parameters is the third best choice for important relative errors. As for 50% of attackers, RVivaldi always outperforms Vivaldi and the age bonus coefficient is not important.

RVivaldi can improves the accuracy of coordinates with repulse attacks. If the number of attackers is unknown, we recommend to choice $\gamma = 0.1$ and $h = 10$. Indeed, this choice is always the best with less than 50% of attackers and the best for low relative errors in presence of 50% of attackers. If the number of attackers increases, this choice is always within the four best choices. We propose to set c_a to 0.01.

In presence of repulse attack, the best choice is $\gamma = 0.1$, $h = 10$ and $c_a = 0.01$. Indeed, this choice is the best tradeoff between good coordinates in presence of few or many attackers.

Conclusion: With the simulations presented above, we see that it is unfortunately not possible to choose a combination of parameters such that RVivaldi always offers its best results. If the attacks and the number of attackers are known, the parameters can be chosen such that RVivaldi gives the more accurate coordinates. On the contrary, if the attacks and their number are not predictable, **we recommend to set $\gamma = 0.1$ and $h = 10$.** Indeed, this choice of γ and h offers the best results most of the time and is always in the top four best choices of parameters. An other property in favour of this choice is that RVivaldi always outperforms Vivaldi for this combination of parameters. In our simulations, the age bonus coefficient has no impact on the accuracy of the coordinates. Thus, **we propose to set c_a to** 0.01.

5.4.3 RVivaldi: The improvements

The previous section described how to choose the best combination of parameters γ, h and c_a. In this section, we analyze in depth the improvement due to the reputation model for the best choice of parameters presented above ($\gamma = 0.1$, $h = 10$ and $c_a = 0.01$).

To determine the improvement of RVivaldi, we first compare the curves of the CDFs of the relative errors of Vivaldi and RVivaldi. These curves are obtained for good nodes – i.e., nodes that do not cheat – after 4000 iterations. Meanwhile, the system converges (i.e., when the coordinates are stable) after about 1800 iterations but we want to see if the reputation could lead to oscillations. The "linespoints" curves are the CDFs obtained for Vivaldi (no reputation) and the "points" curves are the CDFs for RVivaldi. The results are always compared with a reference curve that indicates a random choice of coordinates. The curves of CDFs permit to have an overview of the improvement of coordinates while using RVivaldi instead of Vivaldi. The horizontal axis presents the relative errors and the vertical axis their cumulative distribution in the system.

Secondly, we show the evolution of the reputation ratio which is the ratio of the average relative error during an attack when the reputation is enabled with the average relative error during the same attack when the reputation is disabled. The evolution of the ratio with the time indicates how the system behave during the computation of the coordinates.

We first analyze random attacks. After, we study the behaviour of RVivaldi with constant attacks. Next, we depict the protection of RVivaldi against same attacks. Finally, we see how the system behaves when the attackers perform repulse attacks.

Random attacks:

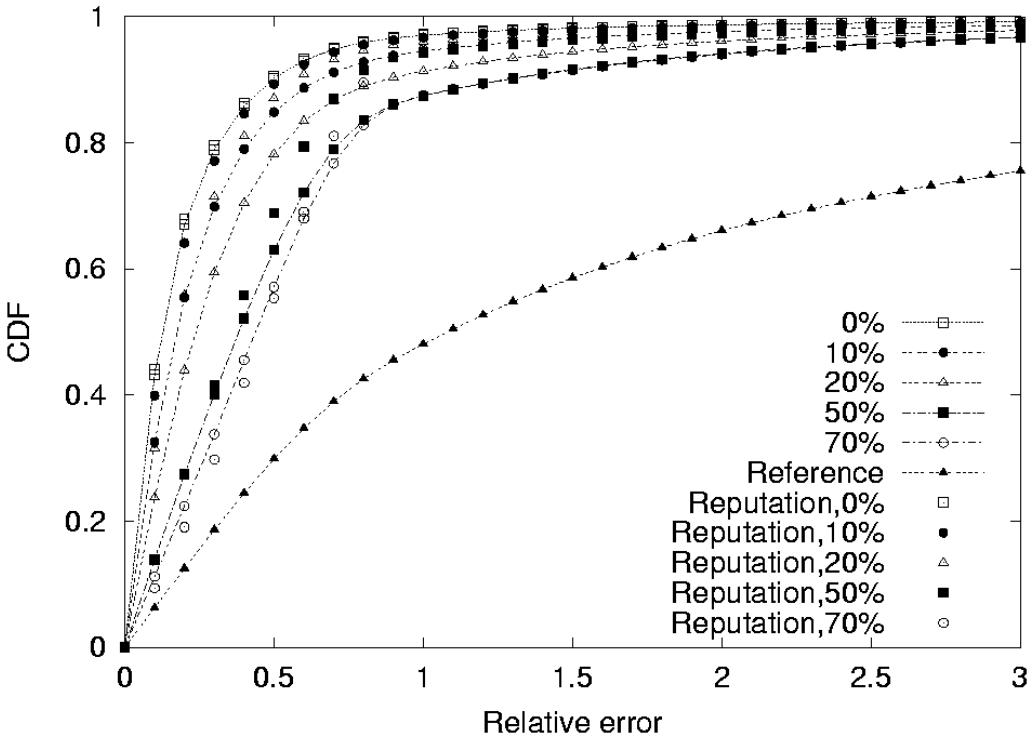


Figure 5.20: CDF of relative errors in RVivaldi for $\gamma = 0.1$, $h = 10$ and $c_a = 0.01$ in presence of random attacks

In Fig. 5.20, it is clear that RVivaldi outperforms Vivaldi for random attacks. The use of RVivaldi in presence of only 10% of attackers permits to obtain the same accuracy than Vivaldi coordinates in absence of attackers. Moreover, the impact of 20% of attackers on RVivaldi is less than the impact of 10% of attacker with Vivaldi. The best improvement of coordinates is observed for relative errors in between 0.25 and 0.75. This result can be explained by the fact that the impact of the random attacks on Vivaldi is particularly observed within this interval (i.e., the difference between the CDFs in absence of attacks and in presence of attacks within this interval is more important than below or above). We observe that the impact of the attacks for relative errors of 2 or more is limited (less than 10% for 70% of attackers). As a results, the improvement due to the reputation is also limited. In other words, important relative errors are not the results of a random attack but the results of the limitations of Vivaldi. Indeed, on average, Vivaldi provides accurate coordinates (less than 20% of the nodes have a relative error above 0.35 in absence of attackers) but some nodes may present important relative errors above 2 (less than 4% of the nodes have a relative error above 2). The reputation model does not improve the accuracy of Vivaldi in absence of attackers. We observe that RVivaldi is better at improving the nodes with low relative error than to limit the number of nodes with important relative errors. RVivaldi also works better for few number of random attackers than for an important portion of misbehaving nodes.

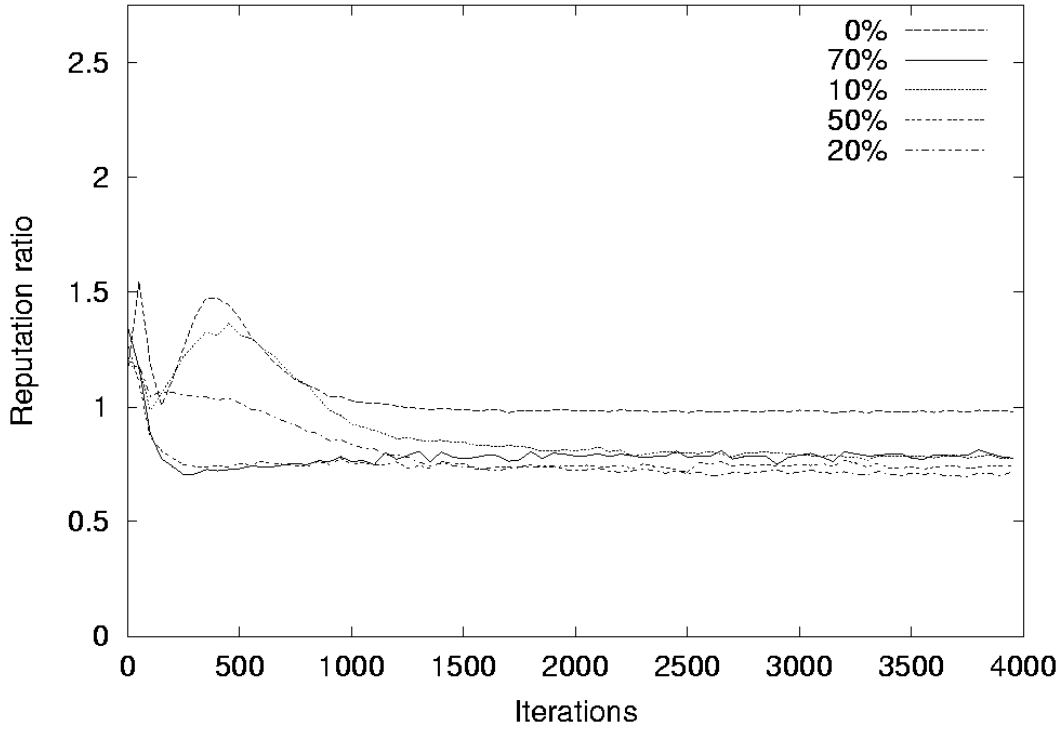


Figure 5.21: Reputation ratio for $\gamma = 0.1$, $h = 10$ and $c_a = 0.01$ in presence of random attacks

Fig. 5.21 clearly shows that RVivaldi outperforms Vivaldi even when the system is not stabilized. Indeed, after only 900 iterations, the reputation ratio is always lower than 1 in presence of attackers. Before, RVivaldi is worst than Vivaldi for 10% and 20% of attackers and better than Vivaldi for 50% and 70% of attackers. In other words, in presence of few attackers, RVivaldi begins by giving innaccurate coordinates compared to Vivaldi but converges to better coordinates than Vivaldi. On the contrary, when the number of attackers increases, RVivaldi is able to rapidly propose more accurate coordinates than Vivaldi. The worst results are obtained in absence of attackers, but the ratio is around but lower than 1 which means that RVivaldi does not improves the accuracy of Vivaldi in absence of attacker but does not alter the accuracy of the prediction. On the contrary, the best results are obtained for 20% of attackers and the ratio converges to more or less 0.75 whatever the number of attackers. A value of 0.75 means than RVivaldi presents an average relative errors 25% lower than Vivaldi for the same attack, we say that *RVivaldi is 25% better than Vivaldi* for random attacks.

Constant attacks:

Fig. 5.22 depicts the evolution of the CDF in presence of constant attacks. RVivaldi always outperforms Vivaldi for constant attacks. Once again, RVivaldi improves the number of nodes with low relative errors and has more difficulties to reduce the number of nodes with high relative errors. Nevertheless, the last observation must be compared with the fact that RVivaldi limits the impact of malicious nodes and that important relative errors are the result of the inherent inaccuracies in coordinate systems. In constant attacks, RVivaldi works better in presence of many malicious nodes than in presence of few attackers: RVivaldi for 70% of attackers offers a better accuracy than Vivaldi for only 50% of misbehaving nodes which can be interpreted as virtual reduction of 20% virtual of the number of attackers. In presence of

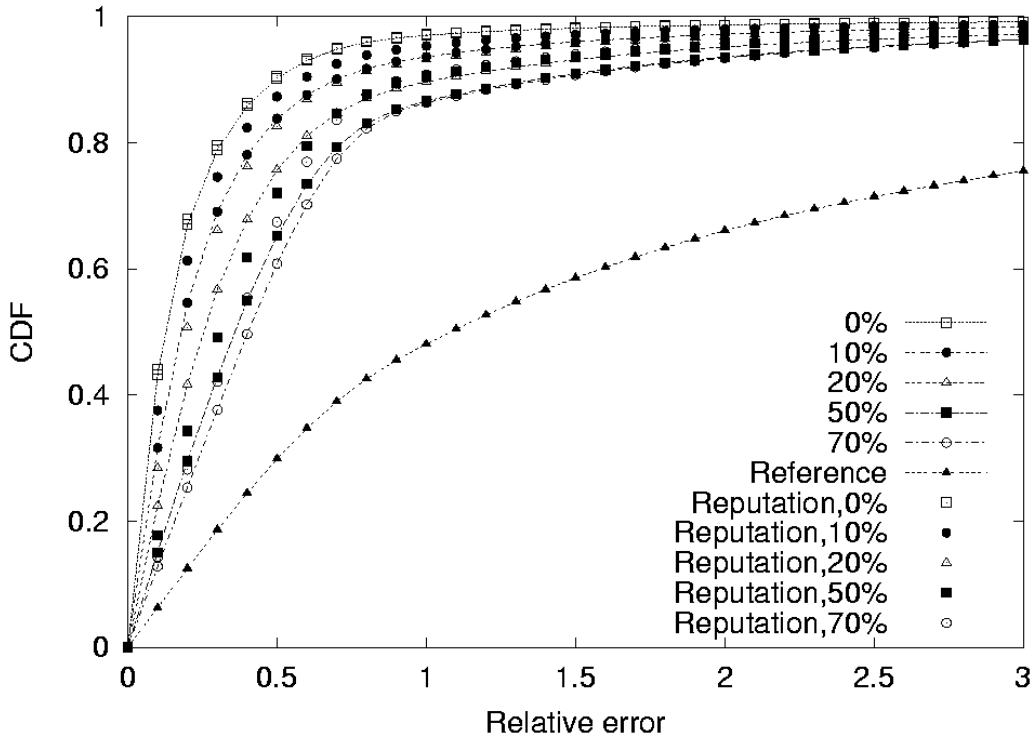


Figure 5.22: CDF of relative errors in RVivaldi for $\gamma = 0.1$, $h = 10$ and $c_a = 0.01$ in presence of constant attacks

only 10% of malicious nodes, RVivaldi is not able to give the same results than Vivaldi in absence of attackers as observed for random attacks.

In Fig. 5.23, we see that RVivaldi outperforms Vivaldi in presence of constant attacks. Indeed, the reputation ratio is below 1 when the system has converged. It is interesting to observe that RVivaldi proposes better coordinates than Vivaldi after less than 200 iterations except for 10% of attackers or less. In presence of attackers, the best results are obtained for 20% of attackers and the reputation ratio is around 0.8 in presence of attackers. Fortunately, in absence of attackers, even if RVivaldi does not really outperforms Vivaldi, the ratio is just below 1. RVivaldi is 20% better than Vivaldi for constant attacks.

Same attacks:

The results obtained for same attacks are presented in Fig. 5.24. Vivaldi is more sensible to this sort of attack than to constant or random attacks. Fortunately, RVivaldi outperforms Vivaldi even for same attacks. We observe above that RVivaldi does not improve the accuracy of coordinates for important relative errors in random and constant attacks because theses inaccuracies are not the results of the attacks. In same attacks, the situation is different and the impact of attacks is equally represented along the relative errors. Indeed, the impact of the attack is more important than the impact of the inherent inaccuracies of Vivaldi. As a result, RVivaldi equally improves the accuracy of coordinates for different relative errors. However, the CDF is only improved by less than 10% and the coordinates for 70% of attackers are still worst than a choice of random coordinates for nodes. The number of attackers does not have a significant impact on the improvement of the accuracy due to the reputation model. How-

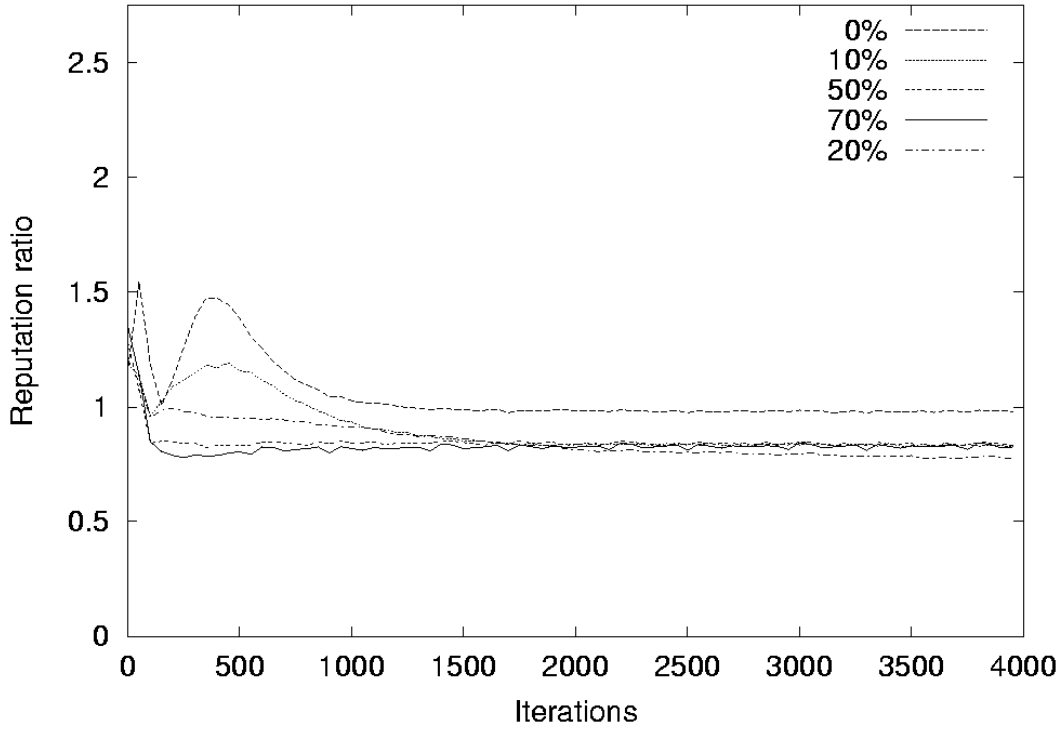


Figure 5.23: Reputation ratio for $\gamma = 0.1$, $h = 10$ and $c_a = 0.01$ in presence of constant attacks

ever, the performances of the coordinate system fastly decrease when the number of attackers increases. For 10% of attackers, only 20% of attackers have a relative error higher than 0.5. With 50% of attackers, this portion reaches 57%. Worst, up to 75% of the nodes present a relative error higher than 0.5 when 70% of attackers are in the system.

Fig. 5.25 depicts the evolution of the reputation ratio with time. We first observe that RVivaldi outperforms Vivaldi when the system has converged. The best results are obtained for 20% of attackers and, once again, the worst are observed in absence of attackers. The converged reputation ratio is about 0.8 and thus RVivaldi is 20% better than Vivaldi. Before the convergence, it is interesting to see that RVivaldi massively improves the accuracy of Vivaldi and proposes a ratio below 0.5. However, the coordinates are not stable and the points moves towards stable coordinates with worse relative error. More interesting, for 70% of attackers, the reputation ratio is the lowest during the transient phase of the computation. On the contrary, the results obtained for 70% of attackers are the worst when the system has converged (except in absence of attackers). On the contrary, the worst results are obtained for 10% of attackers at the beginning, but become the second better after convergence of the system. This observation can be explained by the fact that when only few attackers are in the system, the system behaves like Vivaldi and the impact of RVivaldi on the transient phase is limited. On the contrary, if many malicious nodes are present, the system is far from working like Vivaldi and RVivaldi can strongly influence the transient phase.

Repulse attacks:

As we have already seen, Vivaldi is more sensible to repulse attacks than to same attacks. Fortunately, in Fig. 5.26 we could observe than RVivaldi spectacularly outperforms Vivaldi

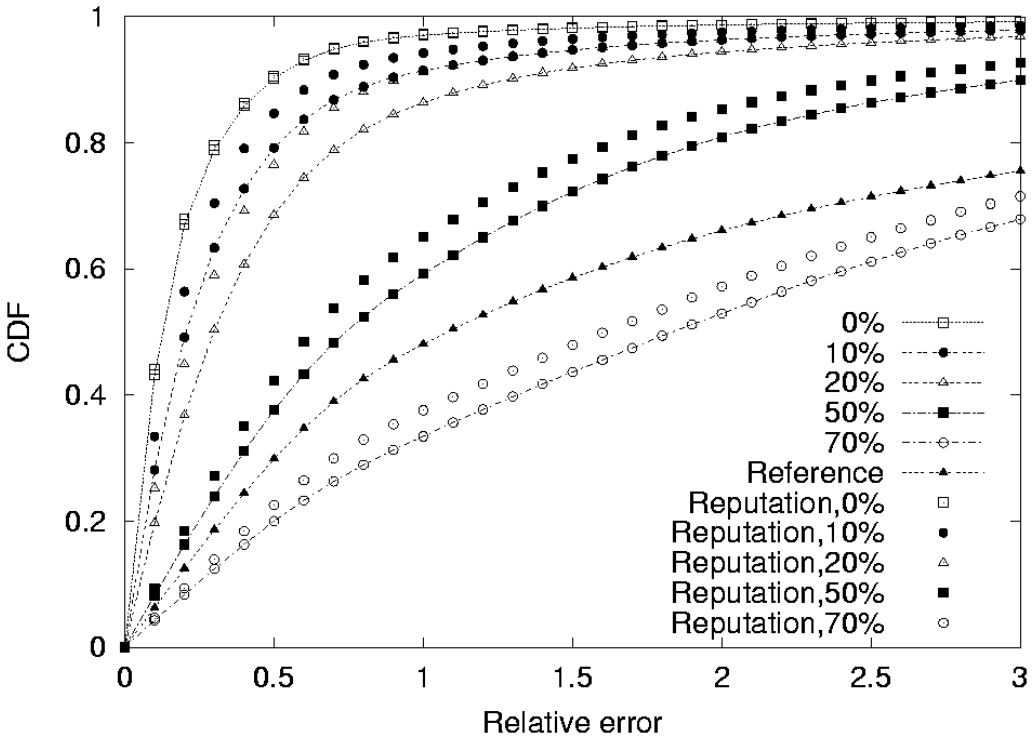


Figure 5.24: CDF of relative errors in RVivaldi for $\gamma = 0.1$, $h = 10$ and $c_a = 0.01$ in presence of same attacks

compared to the results obtained for the other attacks. As observed for same attacks, the impact of a repulse attack is equally distributed among the relative errors. As expected, the impact of RVivaldi on the relative errors is the same for small and high errors. RVivaldi works better in presence of only few attackers (less than 20%) than in presence of numerous attackers. While more than half of the nodes in Vivaldi have a relative error larger than 0.5, only 16% of the nodes present the same characteritics with RVivaldi. In other words, RVivaldi reduces the number of nodes with a relative error higher than 0.5 by 34%. The same observation for 50% of attackers shows that 70% of nodes in Vivaldi have a relative error higher than 0.5. This proportion is reduced to 58% when RVivaldi is used. The improvement is not as important as for 10% of attackers, but 12% more nodes presents relative errors lower than 0.5. The improvement for 70% of attackers is around 10%. It is interesting to see than Vivaldi gives worse coordinates than a random choice of coordinates for 50 of 70% of repulse attackers. On the other hand, RVivaldi gives better coordinates than random ones for the same number of attackers. RVivaldi also allows to have approximatively the same accuracy in presence of few repulse attackers (10%) than RVivladi does for random and constant attacks even though Vivaldi presents worst results for 10% of repulse attackers than it presents in presence of 70% of random or constant attackers.

We saw in Fig. 5.26 that RVivaldi is particularly efficient for repulse attacks. Fig. 5.27 confirms the results. When the system is stable, the reputation ratio is 0.25, 0.35, 0.55 and 0.62 for 10%, 20%, 50% and 70% of attackers respectively. In other words, the average relative error is 75% lower for 10% of attackers in RVivaldi than in Vivaldi. The improvement decreases to 38% for 70% of attackers. The results are good compared to the results obtained for the other types of attacks but there is an important drawback. Indeed, for the previous attacks, the

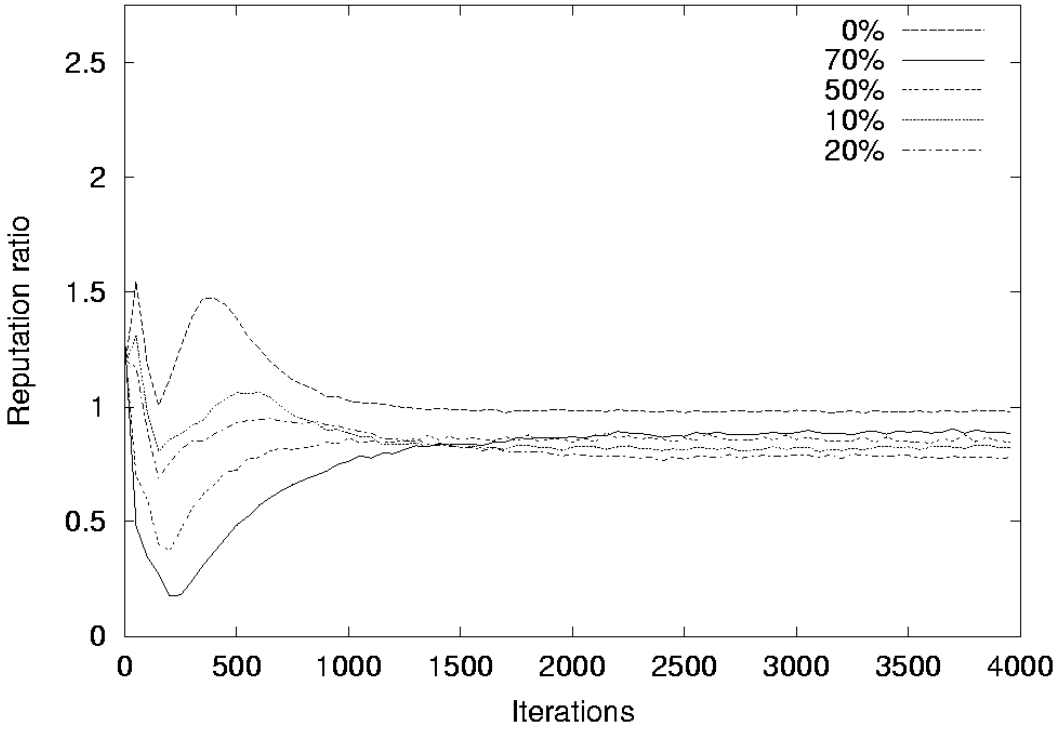


Figure 5.25: Reputation ratio for $\gamma = 0.1$, $h = 10$ and $c_a = 0.01$ in presence of same attacks

reputation ratio is always between 0.5 and 1.5. For repulse attacks, the results can be below 0.5 when the system has converged. But the reputation ratio can also be higher than 2.5 during the transient phase. Such a ratio means that the coordinates proposed by RVivaldi before convergence can be more than 2.5 times worst than those proposed by Vivaldi. In other types of attacks, the worst coordinates are only 1.5 times worst than Vivaldi during the transient phase. In presence of 50% or more misbehaving nodes, RVivaldi does not present the same problem. The peaks to an important relative ratio are obtained for 10% and 20% of attackers. The coordinates of nodes in presence of 10% or 20% of attackers are the best when the system has converged but are the worst during the transient phase. We also observe important changes in the reputation ratio during the transient phase. That means that the nodes are not stable at all and always change their coordinates. Finally, in presence of repulse attackers, RVivaldi must be used with care. Indeed, if the system is stabilized, the improvement compared to Vivaldi is important and RVivaldi can be up to 75% better than Vivaldi. But, if the system has not yet converged, the coordinates can be really worse than Vivaldi and invalidate any decision based on the predicted distance.

Conclusion: RVivaldi is a valuable tool that allows to limit the impact of attacks on Vivaldi. With the choice of $\gamma = 0.1$, $h = 10$ and $c_a = 0.01$, the predicted distances in presence of malicious nodes are always better with reputation than without. However, when the coordinates are not stabilized, the predicted distances can be worse than those proposed by Vivaldi. This is not surprising as RVivaldi is network coordinate system. Indeed, most of the time, the coordinates proposed by NCS are bad during their transient phase and become good once they have stabilized. Moreover, de Launois observed that to propose more accurate coordinates, a coordinate system like Vivaldi takes longer time to converge [11]. This is exactly what we observe: RVivaldi gives better coordinates than Vivaldi but takes more time to converge.

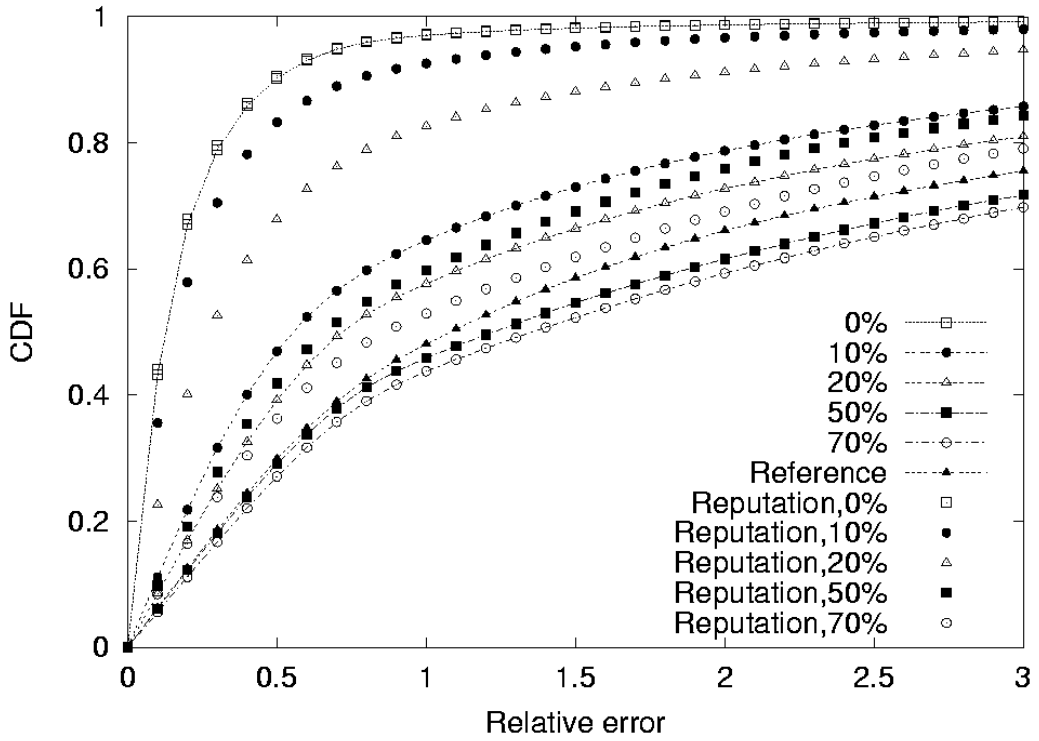


Figure 5.26: CDF of relative errors in RVivaldi for $\gamma = 0.1$, $h = 10$ and $c_a = 0.01$ in presence of repulse attacks

5.5 Number of surveyors

The number of surveyors should depends on the number of malicious nodes in the system. Unfortunately, in practice, it is impossible to know the number of attackers. Thus, the number of surveyors must be the best tradeoff between the quality of the reputation estimation and the lightness of the protocol for any number of malicious nodes. Indeed, if many surveyors are attached to a single node, the reputation estimation is improved. But if the number of surveyors increases, the load on the network increases and the number of nodes monitored by a single surveyor also increases, thus, more computation time is consumed.

To evaluate the number of surveyors needed, we compute the average relative error after convergence for different number of surveyors and others parameters set to their optimal values ($\gamma = 0.1$, $h = 10$ and $c_a = 0.01$). We then compute the average of the average relatives errors for each test with the same number of surveyors. The number of surveyors that presents the minimum error is the best choice for the number of surveyors.

Unfortunately, we do not had time to test the best number of surveyors but the evolution of the error with the number of surveyors can be of two type. Either the error strictly decreases with the augmentation of the number of surveyors or the error decreases and reaches a minimum and then increases. Fig. 5.28(a) shows the evolution of the error with the number of surveyors for the first possibility and Fig. 5.28(b) depicts the evolution for the second possibility. In the second case the number of surveyors to choose is the one that gives the minimum. For a strictly decreasing error, the choice is hared. Indeed, in such a situation, the best solution is

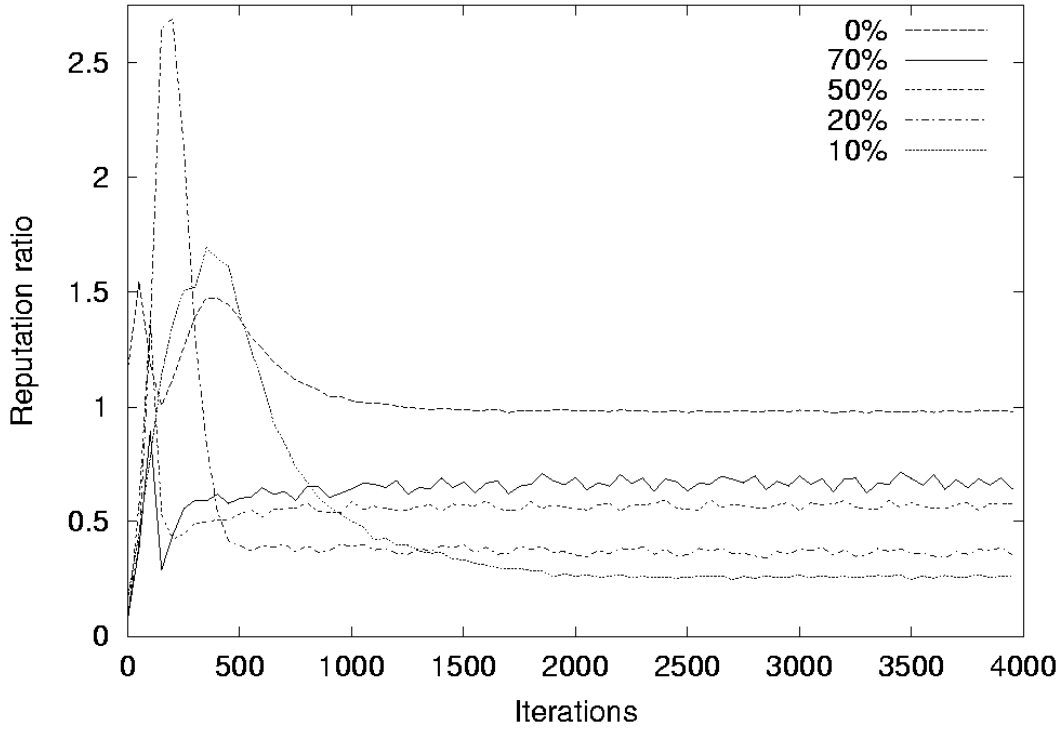
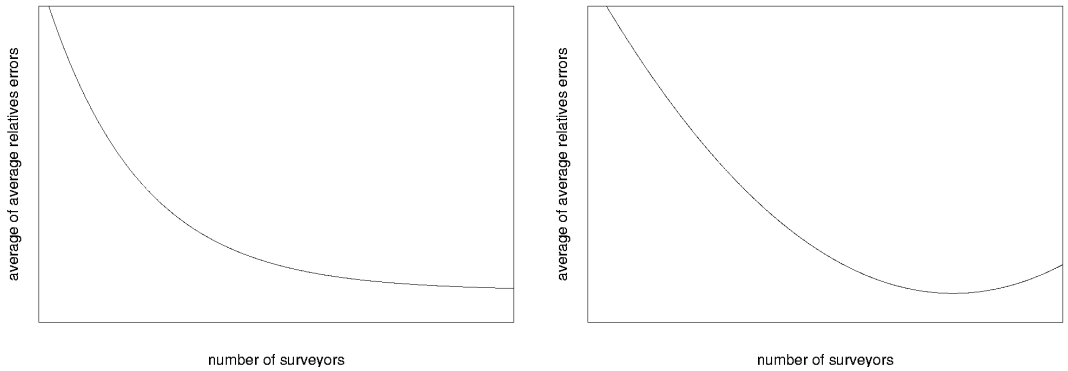


Figure 5.27: Reputation ratio for $\gamma = 0.1$, $h = 10$ and $c_a = 0.01$ in presence of repulse attacks

to set every nodes as surveyors of every nodes which is not acceptable for scalability reasons. A better choice is to choose the number of surveyors such that the diminution of the error due to the addition of one surveyor is less interesting than the cost of adding one surveyor. If it is impossible to define the cost of the addition of a surveyor per node, we can increases the number of surveyors till the diminution of the errors reaches a given threshold.



(a) Strictly decrease of the average of average relatives errors with the number of surveyors

(b) Average of average relatives errors with a global minimum

5.6 Conclusion

In this chapter, we evaluated the reputation model on Vivaldi using RVivaldi. We first determine the best values for the 3 tuning parameters of the reputation model. The experiments show that the best combination is $\gamma = 0.1$, $h = 10$ and $c_a = 0.01$. The experiments for different attacks show that the reputation model limits the impact of attacks on Vivaldi. The results confirm that the reputation model is a valuable extension to network coordinates system in unsecured environments. In presence of attackers, RVivaldi always outperforms Vivaldi for our choice of parameters. In this chapter we also saw that the improvement due to the addition of the reputation depends on the type of attack and the number of malicious nodes.

6 Conclusion and further work

The goal of this thesis was to propose a network coordinate system able to resists to attacks.

Network Coordinate Systems have been proposed to allow hosts to estimate delays between nodes without having to contact them first. Nodes compute their coordinates into a virtual geometric space such that the distance from itself to any host predicts the latency to that node. The main idea of coordinates-based system is that every host maintains coordinates. The Internet is represented as a space where each point with a particular coordinates is a node. The more accurate the coordinate space model, the more accurate the prediction of the distances. For short, a coordinates-based system must propose a mapping of Internet hosts to a virtual space such that the distances in this last space are as close as possible to the actual distances between Internet hosts. In network coordinate systems, the coordinates are computed using the coordinates of a set of well-chosen nodes and the distance to them. Coordinate systems, such as Vivaldi or NPS, might be used in various applications where the notion of proximity, expressed as network delay or RTT, is used.

There is two families of network coordinate systems: (*i*) Landmarks-based NCS and (*ii*) decentralize NCS. In landmarks-based NCS, an architecture of well-known nodes is maintained and normal nodes compute their coordinates based on the coordinates of the landmarks. On the contrary, on decentralize network coordinate systems, there is no distinction between nodes and any node can be used as a landmark. During the survey, we observed that nodes in coordinate systems rely on what their references pretends.

It has been demonstrated that network coordinate system are valuable tools but, unfortunately, it has been shown that such systems are sensible to attacks. Indeed, a malicious node can lie about its coordinates and, as a consequence, deform the coordinate space. On network coordinate systems, the attacks can come from the outside or from the inside. In inside attacks, the malicious nodes are trusted nodes that give bad information. Typically, a misbehaving node gives fake coordinates and thus deforms the coordinates space. While it is simple to protect a system against external attacks, the large-scale nature of coordinate systems make them impossible to protect perfectly from internal attacks. Nevertheless, it is possible to suspect if a node lye and thus limit the capability of a bad node to alter the coordinates of a good node.

To protect network coordinate systems against attacks, we propose a reputation model. In this model, the nodes are monitored by other nodes: The surveyors. When a node changes its coordinates, it contacts a special certification agent: The RCA. The RCA then computes how correct the coordinates are based on the observations made by the surveyors. This metric is evaluated by the reputation. The reputation is a value that indicates the probability that a node does not lie (i.e., a value near 0 means that the probability the node lies is important. On the contrary, a value near 1 means that the node seems to be honest). We propose to

use the reputation during the computation of the coordinates. Indeed, if a node has a high reputation, one could use its information without particular attention. On the contrary, if a node has a low reputation, information given by the node must be taken with care. In addition to coordinates computation, the reputation can be used to select the closer node. To this aim, we propose the score. In traditional network coordinate systems, the closer node is the node with the lower predicted distance. With the score, the distance is still minimized but nodes with a good reputation are favoured.

The model of reputation is virtually compatible with every network coordinate systems and we propose RVivaldi, a reputation approach of Vivaldi. To secure Vivaldi with the reputation model, it is only required to modify the timestep function (δ) of Vivaldi and implement the network coordinate system above the reputation model framework.

The reputation model was tested on Vivaldi using RVivaldi. This model depends on three parameters, γ, h and c_a. γ defines the weight given to the last experience. h is the number of previous experiences used to compute the reputation. Finally, c_a limits the impact of new nodes in the system in favour of more aged nodes. The experiments show that the best choice for the parameters is $\gamma = 0.1$, $h = 10$ and $c_a = 0.01$. With the experiments, we show that the reputation model limits the impact of attacks on Vivaldi. The results confirm that the reputation model is a valuable extension to network coordinates system in unsecured environments. The main contribution of the experiments is to indicate that, in presence of attackers, RVivaldi always outperforms Vivaldi for our choice of parameters.

The purpose of this work was to propose a new approach to secure network coordinate systems. After a study of the attacks on network coordinate systems, we proposed an approach based on the reputation. The reputation gives an information on the probability a node in a network coordinate system is malicious or not. With different simulations, we showed that the reputation model we propose limits the impact of attacks on network coordinate systems.

6.1 Further work

In the near future, we aim at validating RVivaldi in a real environment, such as PlanetLab [40]. We would also be interested by testing the reputation on other coordinate systems like NPS or BBS and see the impact of the reputation on these systems.

Unfortunately, the solution we propose rely on a centralized node, the RCA. For the next future, a solution must be found to decentralize the RCA and thus improve the scalability of reputation-based coordinate systems. In addition, we proposed a methodology to determine the best number of surveyors in Sec. 5.5 without testing it. In the next future, some experiments have to be done to validate this methodology.

Moreover, it could be interesting to formally demonstrate the correctness of the reputation model and maybe propose a statistical model to determine the best choice of the different parameters involving in the reputation computation.

Finally, the reputation model has been proposed to compute more accurate coordinates in presence of malicious nodes and to detect the most probable closer node. The reputation could also be used for monitoring. Indeed, a sudden change of reputation could be the sign of a topology change or the result of an attack.

Bibliography

[1] M. Pias, J. Crowcroft, S. Wilbur, T. Harris, and S. Bhatti, "Lighthouses for scalable distributed location," in *Proc. 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003.

[2] F. Dabek, R. Cox, K. Kaashoek, and R. Morris, "Vivaldi, a decentralized network coordinated system," in *Proc. ACM SIGCOMM*, August 2004.

[3] T. S. E. Ng and H. Zhang, "A network positioning system for the internet," in *Proc. USENIX Annual Technical Conference*, June 2004.

[4] M. Kaafar, L. Mathy, T. Turletti, and W. Dabbous, "Virtual networks under attack: Disrupting internet coordinate systems," in *Proc. ACM CoNEXT*, December 2006.

[5] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proc. ACM SIGCOM*, August 2001.

[6] S. Ratnasamy, P. Francis, M. Handley, R. Kerp, and S. Shenker, "A scalable content-addressable network," in *Proc. ACM SIGCOMM*, August 2001.

[7] P. Francis, S. Jamin, V. Paxson, L. Zhang, D. F. Gruniewicz, and Y. Jin, "An architecture for a global internet host distance estimator service," in *Proc. IEEE INFOCOM*, March 1999.

[8] T. Ng and H. Zhang, "Predicting internet network distance with coordinates-based approaches," in *Proc. IEEE INFOCOM*, June 2002.

[9] Y. Shavitt and T. Tankel, "Big-bang simulation for embedding network distances in euclidean space," in *Proc. IEEE INFOCOM*, March 2003.

[10] M. Costa, M. Castro, A. Rowstron, and P. Key, "Pic: Practical internet coordinates for distance estimation," Microsoft Research, Tech. Rep. MSR-TR-2003-53, September 2003.

[11] C. de Launois, S. Uhlig, and O. Bonaventure, "Scalable route selection for IPv6 multihomed sites," in *Proc. IFIP Networking*, May 2005.

[12] R. Prasad, M. Murray, C. Dovrolis, and K. Claffy, "Bandwidth estimation: metrics, measurement techniques, and tools," *IEEE Network*, vol. 17, pp. 27–35, November 2003.

[13] "Azureus," February 2007, http://azureus.sf.net.

[14] "Allpeers," April 2007, http://www.allpeers.com.

[15] J. Ledlie, P. Gardner, and M. Seltzer, "Network coordinates in the wild," in *Proc. Symposium on Networked Systems Design and Implementation (NSDI)*, April 2007.

[16] M. Kaafar, L. Mathy, T. Turletti, and W. Dabbous, "Real attacks on virtual networks: Vivaldi out of tune," in *Proc. SIGCOMM workshop LSAD 2006*, September 2006.

[17] B. Donnet and O. Bonaventure, "On the vulnerability of internet measurements," Tech. Rep., April 2007.

[18] D. Saucez, B. Donnet, and O. Bonaventure, "A reputation-based approach for securing vivaldi embedding system," in *Proc. 13th EUNICE Open European Summer School and IFIP TC6.6 Workshop on Dependable and Adaptable Networks and Service*, Twente, The Netherlands, July 2007.

[19] K. P. Gummadi, S. Saroiu, and S. D. Gribble, "King: Estimating latenccy between arbitrary internet end hosts," in *Proc. 2nd USENIX/ACM SIGCOMM Internet Measurement Wokshop (IMW)*, November 2002.

[20] M. Goodrich and R. Tamassia, *Data Structures and Algorithms in Java*, 3rd ed. Wiley, 2004, ch. 12.

[21] G. Strang, *Introduction to Linear Algebra*, 3rd ed. Wellesley-Cambridge Press, March 2003.

[22] H. Young, R. Freedman, T. Sandin, and A. Ford, *Sears and Zemansky's University Physics With Modern Physics*, ser. Physics. Addison-Wesley, 2000.

[23] S. J., R. Norvig, and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed., ser. Artificial Intelligence. Prentice Hall, 2002.

[24] H. Zheng, E. Lua, P. M., and T. Griffin, "Internet routing policies and round-trip times," in *Proc. Passive Active Measurement (PAM)*, March 2005.

[25] E. Lua, T. Griffin, M. Pias, H. Zheng, and J. Crowcroft, "On the accuracy of embeddings for internet coordinate systems," in *Proc. Internet Measurement Conference (IMC)*, October 2005.

[26] C. Jin, H. Wang, and K. Shin, "Hop-count filtering: An effective defence against spoofed DoS traffic," in *Proc. of the 10th ACM International Conference on Computer and Communications Security (CCS)*, 2003.

[27] A. Tanenbaum, *Computer Networks*, 4th ed. Prentice Hall, 2003, ch. 5.

[28] A. Jϕsang, R. Ismail, and C. Boyd, "A survey of trust and reputation systems for online service provision," *Decision Support Systems*, vol. 43, no. 2, pp. 618–644, March 2007.

[29] L. Rasmusson and S. Janssen, "Simulated social control for secure internet commerce," in *Proc. ACM New Security Paradigms Workshop*, April 1996.

[30] M. Kinateder and K. Rothermel, "Architecture and algorithms for a distributed reputation system," in *Proc. 1st Conference on Trust Management*, May 2003.

[31] A. Jϕsang, "A logic for uncertain probabilities," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 9, no. 3, pp. 279–311, June 2001.

[32] A. Twigg, "A subjective approach to routing in p2p and ad hoc networks," in *Proc. 1st Conference on Trust Management (iTrust)*, May 2003.

[33] M. Gupta, P. Judge, and M. Ammar, "A reputation system for peer-to-peer networks," in *Proc. 13th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, June 2003.

[34] M. Kaafar, L. Mathy, C. Barakat, K. Salamatian, T. Turletti, and W. Dabbous, "Securing internet coordinate embedding systems," in *Proc. ACM SIGCOMM*, August 2008.

[35] B. Yu, M. Singh, and K. Sycara, "Developping trust in large-scale peer-to-peer systems," in *Proc. 1st IEEE Symposium on Multi-Agent Security and Survivability*, August 2004.

[36] D. Mills, "Network time protocol (version 3) specification, implementation and analysis," Internet Engineering Task Force, RFC 1305, March 1992.

[37] P. Delsarte and B. Macq, "Ingi 2348: Théorie de l'information et du codage," October 2003, université Catholique de Louvain (UCL).

[38] T. Aura, "Cryptographically generated addresses (CGA)," 2004, draft-ietf-send-cga-05.txt.

[39] W. Feng, E. Kaiser, and A. Luu, "The desing and implementation of network puzzles," in *Proc. IEEE INFOCOM*, March 2005.

[40] "Planetlab," May 2007, https://www.planet-lab.org/.

A Acronyms

AP : Address Prefix
AS : Autonomous System
BBS : Big-Bang Simulation
CAN : Content Addressable Network
CDF : Cummulative Distribution Function
DDoS : Distributed Denial of Service
DHS : Down-Hill Simplex
DoS : Denial of Service
E2E : End-to-End
GNP : Global Network Positioning
HbH : Hop-by-Hop
ICMP : Internet Control Message Protocol
IDMaps : Internet Distance Map Service
IP : Internet Protocol
IPv4 : Internet Protocol version 4
IPv6 : Internet Protocol version 6
ISP : Ineternet Service Provider
NCS : Network Coordinate Systems
NCS : Network Coordinates Systems
NPS : Network Positioning System
PIC : Practical Internet Coordinates
RCA : Reputation Computation Agent
RTT : Round Trip Time
TCP : Transmission Control Protocol
VHV : Variation History Vector
VoIP : Voice over IP

B Source code

This Appendix gives the code of the simulator. For tests or modifications, the files are available at `http://ece.fsa.ucl.ac.be/dsaucez/thesis/simulator/src/`.

The application and its sources is under the following licence (FreeBSD license).

Listing B.1: `Configurator.java`

```
/*-
 * Copyright (c) 2006, 2007
 *  Damien Saucez.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *  This product includes software developed by Damien Saucez.
 * 4. Neither the name of the University nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS
 * BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
 * THE POSSIBILITY OF SUCH DAMAGE.
 */
package simulator;
import java.util.Random;
import java.util.StringTokenizer;

/**
 * The Configuration class permits to configure a simulator based on the
 * content of configuration files. The nodes in the simulator must
 * implement the class VivaldiNode. The nodes collector and the distances
 * collector of the simulator must be initialized and have the correct
 * dimension.
 *
 * @author Damien Saucez
 * @version May 2007
 */
public class Configurator{
  /** File accessor (file abstraction) */
  private FileAccessor fa;

  /**
   * Configure the neighbours of the nodes in the simulator "simu" using
   * the configuration file named "file".
   * The configuration file is composed of two columns. The first contains
   * the identifiers of the nodes and the second is the identifier of the
   * neighbour of this node. The seperation between columns is any number
   * of tabs or spaces.
   * Empty lines and lines beginning with a # are ignored.
   * All the nodes must exist before calling this method.
   *
   * example of file:
   * #neighbours
   * #nid    neigh
   * 0    1636
   * 0    1524
   *
   * 1    1636
   *
   * In this example, nodes 1636 and 1524 are the neighbours of the node 0
   * and node 1636 is the neighbour of node 1.
   *
   * @param simu reference to the simulator
   * @param file name of the neighbour configuration file
   */
  public static void configNeighbours(final Simulator simu,
    final String file){
/* Open the configuration file */
FileAccessor fa = new FileAccessor(file);
if(!fa.openRead()){}

String ln;      /* current line */
StringTokenizer tokenizer; /* tool to divide line in tokens */
String token;    /* current token of the line */
/* read the file line by line */
while((ln = fa.nextLine()) != null){
  if(ln.length() == 0){continue;}

  tokenizer = new StringTokenizer(ln);
  while(tokenizer.hasMoreTokens()){
 token = tokenizer.nextToken();
 try{
   /* ignore comments and empty lines */
   if(token.length() > 0 && token.charAt(0)=='#'){break;}
 }catch(StringIndexOutOfBoundsException e){break;}

 /* node to attach a neighbour to */
 Node src = simu.nc.getNode(Integer.parseInt(token));
 /* neighbour to attach to src */
 Node dst = simu.nc.getNode(Integer.parseInt(tokenizer.nextToken()));
 ((VivaldiNode)src).addNeighbour(dst);
  }
}
/* close the configuration file */
if(!fa.close()){}
 }

 /**
  * Configure the matrix of distances for the nodes in the simulator
  * "simu" using the configuration file named "file".
  * The configuration file is composed of three columns. The first
  * contains the identifiers of the source nodes of the distances, the
  * second is the identifier of destination of the distance and the third
  * column is the distance between the source and the destination. The
  * seperation between columns is any number of tabs or spaces. A negative
  * distance of -1 means that the distance is unknown.
  * Empty lines and lines beginning with a # are ignored.
  *
  * example of file:
  * # rtts
  * #src dst rtt
  * 0    0    0.0
  * 0  1    105.042
  * 0  2  -1.0
  *
  * 1  0    102.10
  *
  * In this example, the distance between the node 0 and itself is 0. The
  * distance between 0 and 1 is 105.042 and the distance between 1 and 0
  * is 102.10. This illustrates the that distances must not been symetric.
  * The distance between 0 and 2 is unknown.
  *
  * @param simu     reference to the simulator
  * @param file     name of the neighbour configuration file
  */
```

```
134     public static void configDistances(final Simulator simu,
135       final String file){
136   /* open the configuration file*/
137   FileAccessor fa = new FileAccessor(file);
138   if(!fa.openRead()){}
139
140   String ln;      /* current line */
141   StringTokenizer tokenizer;  /* tool to divide line in tokens */
142   String token;     /* current token of the line */
143   /* read the file line by line */
144   while((ln = fa.nextLine()) != null){
145     if(ln.length() == 0){continue;}
146
147     tokenizer = new StringTokenizer(ln);
148     while(tokenizer.hasMoreTokens()){
149    token = tokenizer.nextToken();
150
151    try{
152      /* ignore comments and empty lines */
153      if(token.length() > 0 && token.charAt(0)=='#'){break;}
154    }catch(StringIndexOutOfBoundsException e){break;}
155
156    /* source node for the distance */
157    int src = Integer.parseInt(token);
158    /* destination node for the distance */
159    int dst = Integer.parseInt(tokenizer.nextToken());
160    /* distance between src and dst */
161    double rtt = Double.parseDouble(tokenizer.nextToken());
162    simu.dc.setDistance(src, dst, rtt);
163     }
164   }
165   /* close the configuration file */
166   if(!fa.close()){}
167    }
168
169    /**
170     * Configure the attacks performed by the nodes in the simulator "simu"
171     * using the configuration file named "file".
172     * The configuration file is composed of two columns. The first contains
173     * the type of attack and the second the identifier of the node which
174     * performs the attack. The seperation between columns is any number of
175     * tabs or spaces.
176     * Empty lines and lines beginning with a # are ignored.
177     * All the nodes must exist before calling this method.
178     *
179     * The configurator support random, constant, same and repulse attacks.
180     * A constant attack is given with the keyword "const"
181     * A same attack is given with the keyword "same"
182     * A random attack is given by the keyword "rand"
183     * A repulse attack is given by the keyword "repuls"
184     * The attackers have a local error set to 0.1. Constant attackers have
185     * coordinates randomly chosen in [0;500] in the 3-dimensions space. The
186     * target for repulse attack is randomly chosen in [0;1000] and the
187     * target of the node is (10000 + nid) where nid is the identifier of
188     * the node.
189     *
190     * example of file:
191     * repuls  1654
192     * same    1422
193     * repuls  1520
194     *
195     * In this example, nodes 1520 and 1654 performs repulse attacks and
196     * 1422 is a same attacker.
197     *
198     * @param simu     reference to the simulator
199     * @param file     name of the neighbour configuration file
200     */
201     public static void configAttackants(final Simulator simu,
202       final String file){
203   /* open the configuration file */
204   FileAccessor fa = new FileAccessor(file);
205   if(!fa.openRead()){}
206
207   String ln;                        /* current line */
208   StringTokenizer tokenizer;        /* tool to divide line in tokens */
209   String token;                     /* current token of the line */
210   Random rand = new Random(12345);/* random number generator */
211   /* read the file line by line */
212   while((ln = fa.nextLine()) != null){
213     if(ln.length() == 0){ continue;}
214
215     tokenizer = new StringTokenizer(ln);
216     while(tokenizer.hasMoreTokens()){
217    token = tokenizer.nextToken();
218
219    try{
220      /* ignore comments and empty lines */
221      if(token.length() > 0 && token.charAt(0)=='#'){break;}
222    }catch(StringIndexOutOfBoundsException e){break;}
223
224    /* Random attack */
225    if(token.equals("rand")){
226      int nid = Integer.parseInt(tokenizer.nextToken());
227      /* node that performs the attack */
228      Node n = simu.nc.getNode(nid);
229      /* n is a random attacker */
230      n.attack = Lib.Attack.RANDOM;
231
232      ((VivaldiNode)n).e_i = 0.1; /* set local error to 0.1 */
233    }
234    /* Constant attack */
235    else if(token.equals("const")){
236      int nid = Integer.parseInt(tokenizer.nextToken());
237      /* node that performs the attack */
238      Node n = simu.nc.getNode(nid);
239      /* n is a constant attacker */
240      n.attack = Lib.Attack.CONSTANT;
241
242      ((VivaldiNode)n).e_i = 0.1; /* set local error to 0.1 */
243      /* the node has 3D random coordinates between 0 and 500 */
244      n.setCoords((Coord)new EuclideanCoord(new double[]{
245     rand.nextDouble()*500d,
246     rand.nextDouble()*500d,
247     rand.nextDouble()*500d}));
248    }
249    else if(token.equals("same")){
250      int nid = Integer.parseInt(tokenizer.nextToken());
251      /* node that performs the attack */
252      Node n = simu.nc.getNode(nid);
253      /* n is a same attacker */
254      n.attack = Lib.Attack.SAME;
255
256      ((VivaldiNode)n).e_i = 0.1; /* set local error to 0.1 */
257    }
258    /* Repulse attack */
259    else if(token.equals("repuls")){
260      int nid = Integer.parseInt(tokenizer.nextToken());
261      /* node that performs the attack */
262      Node n = simu.nc.getNode(nid);
263      /* n is a repulse attacker */
264      n.attack = Lib.Attack.REPULSION;
265
266      ((VivaldiNode)n).e_i = 0.1; /* set local error to 0.1 */
267      ((VivaldiNode)n).target = new EuclideanCoord(new double[]{
```

```
268      rand.nextDouble()*500d + 500,
269      rand.nextDouble()*500d + 500,
270      rand.nextDouble()*500d + 500});
271
272       n.setCoords((Coord)new EuclideanCoord(
273       new double[]{10000+(nid),10000+(nid),10000+(nid)}));
274     }
275     break;
276      }
277    }
278    /* close the configuration file */
279    if(!fa.close()){}
280     }
281
282     /**
283      * Configure the surveyors of the nodes in the simulator "simu" using the
284      * configuration file named "file".
285      * The configuration file is composed of two columns. The first contains
286      * the identifiers of the nodes and the second is the identifier of one
287      * of its surveyors. The seperation between columns is any number of tabs
288      * or spaces.
289      * Empty lines and lines beginning with a # are ignored.
290      * All the nodes must exist before calling this method.
291      *
292      * example of file:
293      * # surveyors
294      * #nid    neigh
295      * 0   685
296      * 0   1337
297      *
298      * 1   685
299      *
300      * In this example, nodes 685 is a surveyor of node 0 and 1. Node 0 has
301      * also node 1337 has surveyor.
302      *
303      * @param simu     reference to the simulator
304      * @param file     name of the neighbour configuration file
305      */
306     public static void configSurveyrors(final Simulator simu,
307       final String file){
308    FileAccessor fa = new FileAccessor(file);
309    // open the configuration file
310    if(!fa.openRead()){
311    }
312
313    String ln;                          /* current line */
314    StringTokenizer tokenizer;          /* tool to divide line in tokens */
315    String token;                       /* current token of the line */
316    /* read the file line by line */
317    while((ln = fa.nextLine()) != null){
318      if(ln.length() == 0){continue;}
319
320      tokenizer = new StringTokenizer(ln);
321      while(tokenizer.hasMoreTokens()){
322     token = tokenizer.nextToken();
323
324     try{
325       /* ignore comments and empty lines */
326       if(token.length() > 0 && token.charAt(0)=='#'){break;}
327     }catch(StringIndexOutOfBoundsException e){break;}
328
329     int nid = Integer.parseInt(token); /* id of the node */
330     /* id of the surveyor */
331     int surv = Integer.parseInt(tokenizer.nextToken());
332     ReputationNode n = (ReputationNode)(simu.nc.getNode(nid));
333     n.addSurveyor((Surveyor)(simu.nc.getNode(surv)));
334     break;
335       }
336     }
337     /* close the configuration file */
338     if(!fa.close()){}
339      }
340    }
```

Listing B.2: `Coord.java`

```
1   /*-
2    * Copyright (c) 2006, 2007
3    *  Damien Saucez.  All rights reserved.
4    *
5    * Redistribution and use in source and binary forms, with or without
6    * modification, are permitted provided that the following conditions
7    * are met:
8    * 1. Redistributions of source code must retain the above copyright
9    *    notice, this list of conditions and the following disclaimer.
10   * 2. Redistributions in binary form must reproduce the above copyright
11   *    notice, this list of conditions and the following disclaimer in the
12   *    documentation and/or other materials provided with the distribution.
13   * 3. All advertising materials mentioning features or use of this software
14   *    must display the following acknowledgement:
15   *  This product includes software developed by Damien Saucez.
16   * 4. Neither the name of the University nor the names of its contributors
17   *    may be used to endorse or promote products derived from this software
18   *    without specific prior written permission.
19   *
20   * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
21   * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
22   * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
23   * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS
24   * BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
25   * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
26   * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
27   * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
28   * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
29   * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
30   * THE POSSIBILITY OF SUCH DAMAGE.
31   */
32  package simulator;
33
34  /**
35   * Interface for coordinates representation and manipulation.
36   * The size of the table returned by a method or passed as a parameter
37   * gives the dimension of the coordinates
38   *
39   * @author Damien Saucez
40   * @version May 2007
41   */
42  public interface Coord{
43   /**
44    * Retrieve coordinates
45    * @return table of coordinates coponents ((1,2,3) gives the array
46    *     {1.0,2.0,3.0})
47    */
48   public double[] getCoords();
49
50   /**
51    * Set the value to coordinates
52    * @param coords new coordinates
53    */
54   public void setCoords(double coords[]);
55
56   /**
```

```
57     * Sum of two coordinates (this+c2)
58     * @return the sum of current coordinates + c2
59     */
60    public Coord plus(final Coord c2);
61
62    /**
63     * Difference of two coordinates (this-c2)
64     * @return the difference of current coordinates minus c2
65     */
66    public Coord minus(final Coord c2);
67
68    /**
69     * Left scalar product (alpha * this)
70     * @return the left scalar product of the scalar alpha with the
71     *   current coordinates
72     */
73    public Coord leftDotProduct(double alpha);
74
75    /**
76     * Right scalar product (this * alpha)
77     * @return the right scalar product of the current coordinates with
78     *    the scalar alpha
79     */
80    public Coord rightDotProduct(double alpha);
81
82    /**
83     * Norm (||this||)
84     * @return the norm of the current coordinates
85     */
86    public double norm();
87
88    /**
89     * Unitary vector  (this/||this|| = this * 1/||this||)
90     * If the norm is 0, a random unitary vector is returned.
91     * @return the current coordinates normalized to 1
92     */
93    public Coord toUnitary();
94
95    /**
96     * Compare current coordinates to c2
97     * @return true if current coordinates are equals to coordinates of
98     *    c2. Otherwise, a false is returned.
99     */
100   public boolean equals(final Coord c2);
101
102   /**
103    * Clone current object
104    * @return a new object that is a copy of the current
105    */
106   public Coord clone();
107
108   /**
109    * Convert current object to a String representation
110    * @return a string representation of the current object
111    */
112   public String toString();
113 }
```

Listing B.3: DistancesCollector.java

```
1  /*-
2   * Copyright (c) 2006, 2007
3   *  Damien Saucez.  All rights reserved.
4   *
5   * Redistribution and use in source and binary forms, with or without
6   * modification, are permitted provided that the following conditions
7   * are met:
8   * 1. Redistributions of source code must retain the above copyright
9   *    notice, this list of conditions and the following disclaimer.
10  * 2. Redistributions in binary form must reproduce the above copyright
11  *    notice, this list of conditions and the following disclaimer in the
12  *    documentation and/or other materials provided with the distribution.
13  * 3. All advertising materials mentioning features or use of this software
14  *    must display the following acknowledgement:
15  *  This product includes software developed by Damien Saucez.
16  * 4. Neither the name of the University nor the names of its contributors
17  *    may be used to endorse or promote products derived from this software
18  *    without specific prior written permission.
19  *
20  * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
21  * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
22  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
23  * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS
24  * BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
25  * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
26  * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
27  * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
28  * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
29  * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
30  * THE POSSIBILITY OF SUCH DAMAGE.
31  */
32 package simulator;
33
34 /**
35  * A generator/collector of distances between nodes
36  *
37  * @author Damien Saucez
38  * @version May 2007
39  */
40 public interface DistancesCollector{
41   /**
42    * Retrieve the distance from A to B
43    * @param a node id of A
44    * @param b node if of B
45    * @return distance from A to B
46    */
47   public double getDistance(int a, int b);
48
49   /**
50    * Set the distance from A to B to distance
51    * @param a node id of A
52    * @param b node id of B
53    * @param d new value of the distance
54    * @return previous value of the distance from A to B
55    */
56   public double setDistance(int a, int b, double d);
57 }
```

Listing B.4: EuclideanCoord.java

```
1  /*-
2   * Copyright (c) 2006, 2007
3   *  Damien Saucez.  All rights reserved.
4   *
5   * Redistribution and use in source and binary forms, with or without
6   * modification, are permitted provided that the following conditions
7   * are met:
8   * 1. Redistributions of source code must retain the above copyright
9   *    notice, this list of conditions and the following disclaimer.
10  * 2. Redistributions in binary form must reproduce the above copyright
```

```
 *     notice, this list of conditions and the following disclaimer in the
 *     documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *     must display the following acknowledgement:
 *   This product includes software developed by Damien Saucez.
 * 4. Neither the name of the University nor the names of its contributors
 *     may be used to endorse or promote products derived from this software
 *     without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS
 * BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
 * THE POSSIBILITY OF SUCH DAMAGE.
 */
package simulator;

/**
 * Implementation of coordinates in a Euclidean space
 * @author Damien Saucez
 * @version May 2007
 * @see Coord
 */
public class EuclideanCoord implements Coord, java.io.Serializable{
  private double coords[];

  public EuclideanCoord(double coords[]){
 this.coords = coords;
  }

  public double[] getCoords(){
 return this.coords;
  }

  public void setCoords(double coords[]){
 this.coords = coords;
  }

  public Coord plus(final Coord c2){
 double result[] = new double[this.coords.length];
 double coordsc2[] = c2.getCoords();

 for(int i = 0; i < this.coords.length; i++){
   result[i] = this.coords[i] + coordsc2[i];
 }

 return new EuclideanCoord(result);
  }

  public Coord minus(final Coord c2){
 double result[] = new double[this.coords.length];
 double coordsc2[] = c2.getCoords();

 for(int i = 0; i < this.coords.length; i++){
   result[i] = this.coords[i] - coordsc2[i];
 }

 return new EuclideanCoord(result);

  }

  public Coord leftDotProduct(double alpha){
 double result[] = new double[this.coords.length];

 for(int i = 0; i < this.coords.length; i++){
   result[i] = alpha * this.coords[i];
 }

 return new EuclideanCoord(result);
  }

  public Coord rightDotProduct(double alpha){
 return this.leftDotProduct(alpha);
  }

  public double norm(){
 double result = 0d;

 for(int i = 0; i < this.coords.length; i++){
   result += this.coords[i]*this.coords[i];
 }

 return Math.sqrt(result);
  }

  public Coord toUnitary(){
 double result[] = new double[this.coords.length];
 double norm = this.norm();

 for(int i = 0; i < this.coords.length; i++){
   /* if norm is 0, direction is set arbitrary */
   if(norm == 0){
  result[i] = Lib.rand.nextDouble();
   }
   else{
  result[i] = this.coords[i]/norm;
   }
 }

 /* the arbitrary vector is not unitary */
 if(norm == 0){
   /* the norm is never 0 for result */
   return (new EuclideanCoord(result)).toUnitary();
 }

 return new EuclideanCoord(result);
  }

  public boolean equals(final Coord c2){
 double coords_c2[] = c2.getCoords();

 if(coords_c2.length != coords.length){
   return false;
 }

 for(int i = 0; i < coords.length; i++){
   if(coords[i] != coords_c2[i]){
  return false;
   }
 }
 return true;
  }

  public Coord clone(){
 double coords[] = new double[this.coords.length];

 for(int i = 0; i < this.coords.length; i++){
   coords[i] = this.coords[i];
```

```
  }

  return (Coord)new EuclideanCoord(coords);

   }

    public String toString(){
  String result = new String();

  for(int i=0; i < this.coords.length; i++){
    result += this.coords[i]+"\t";
  }

  return result;
   }
}
```

Listing B.5: FileAccessor.java

```
/*-
 * Copyright (c) 2006, 2007
 *  Damien Saucez.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *  This product includes software developed by Damien Saucez.
 * 4. Neither the name of the University nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS
 * BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
 * THE POSSIBILITY OF SUCH DAMAGE.
 */
package simulator;
import java.io.*;

/**
 * Simplifie access to files with an abstracted layer.
 * To open the file, call openRead(). To get the next line in the file,
 * call nextLine(). To close the file, call close().
 * During the read of a file, the errors are not returned to the caller of
 * the method and thus the caller must not implement the catch of
 * exceptions.
 *
 * @author Damien Saucez
 * @version May 2007
 */
public class FileAccessor{

  /** Name of the file */
  private String file;
  /** File accessor */
  private BufferedReader inFile;

  /**
   * Instantiation of a file accessor for file. The file is ready to be
   * opened.
   * @param file name of the file to access
   */
  public FileAccessor(final String file){
 this.file = file;
  }

  /**
   * Open the file for a read only access access
   * @return true if file can be open. Otherwise, a false is returned
   */
  public boolean openRead(){
 try{
   this.file = file;
   this.inFile = new BufferedReader(new FileReader(file));
   return true;
 }
 /* bad file name or not enough rights*/
 catch(FileNotFoundException e){return false;}
 /* access error */
 catch(IOException e){return false;}
  }

  /**
   * Retrieve the next line of the file
   * @return the next line of the file. If there is no more line in the
   *    file or if an error occured, a null is returned
   */
  public String nextLine(){
 try{
   String line = this.inFile.readLine();
   return line;
 }
 catch(IOException e){return null;}
  }

  /**
   * Terminate the access to the file
   * @return true is the file can be closed. Otherwise, a false is
   *     returned
   */
  public boolean close(){
 try{
   this.inFile.close();
   return true;
 }
 /* I/O error */
 catch(IOException e){return false;}
 /* when no open before the close call */
 catch(NullPointerException e){return false;}
  }
}
```

Listing B.6: FlatDistancesCollector.java

```
/*-
 * Copyright (c) 2006, 2007
 *  Damien Saucez.  All rights reserved.
```

```
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *  This product includes software developed by Damien Saucez.
 * 4. Neither the name of the University nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS
 * BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
 * THE POSSIBILITY OF SUCH DAMAGE.
 */
package simulator;
import java.util.StringTokenizer;

/**
 * Implements a basic distances collector
 *
 * @author Damien Saucez
 * @version May 2007
 * @see DistancesCollector
 */
public class FlatDistancesCollector implements DistancesCollector{
  /** Collector of distances */
  protected double distances[][];

  /**
   * Create a distance collector for nbNodes nodes
   * @param nbNodes  number of nodes in the system
   * Note: This distance collector does not embed the construction of its
   * own distance matrix, the matrix must be constructed using
   * setDistance(int,int,int). By default, the distances are all set to 0.
   * Another way to implement a distance collector could be to modify the
   * constructor to directly generate the distance matrix.
   */
  public FlatDistancesCollector(int nbNodes){
 this.distances = new double[nbNodes][nbNodes];
  }

  public double getDistance(int a, int b){
 return distances[a][b];
  }

  public double setDistance(int a, int b, double d){
 double oldD = this.distances[a][b];
 this.distances[a][b] = d;
 return oldD;
  }
}
```


Listing B.7: History.java

```
/*-
 * Copyright (c) 2006, 2007
 *  Damien Saucez.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *  This product includes software developed by Damien Saucez.
 * 4. Neither the name of the University nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS
 * BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
 * THE POSSIBILITY OF SUCH DAMAGE.
 */
package simulator;

/**
 * Store an prepare data in a history of a maximum fixed size. The history
 * can be used to implement the VHV of the formal reputation model.
 * History class computes average and median with amortized constant time
 * to limit the impact on the computation time during a simulation. The
 * history can give the first element of the history or the last but not
 * elements in between. The exponential averaged sum of elements in history
 * are also computed using this class.
 *
 * @author Damien Saucez
 * @version May 2007
 */
public class History{
  /** size of the history */
  private int H;
  /** collection of data */
  private double q[];
  /** cursor to first element */
  private int f;
  /** cursor to last element */
  private int l;
  /** size */
  private int s;
  /** maximum value of the elements in the history */
  private double max;
  /** sum of all the elements in the history */
  private double sum;
  /** the last removed value */
  private double old;
  /** temporar variables to approximate the variance */
  private double s2;
  private boolean exceed;
  /** Exponential averaged sum */
```

```
67     private double expSum;
68     /** age factor */
69     private double age;
70     /** value of (1 - gamma)^{H-1} */
71     private double gammaPowered;
72
73     /**
74      * Instantiate an history of size H
75      * @param H maximum size of the history
76      */
77     public History(int H){
78    this.H = H;
79    this.f = 0;
80    this.l = 0;
81    this.s = 0;
82    this.q = new double[H];
83    this.max = 0;
84    this.sum = 0;
85    this.old = 0;
86    this.s2 = 0;
87    this.exceed = false;
88    this.expSum = 0;
89    this.age = Lib.c_a;
90    /* precompute (1-\gamma)^{H-1} for next computations */
91    this.gammaPowered = Math.pow((1 - Lib.gamma), H-1);
92     }
93
94     /**
95      * Retrieve the oldest element in the history
96      * @return the oldest element
97      */
98     public double first(){
99    return this.q[this.f];
100    }
101
102    /**
103     * Retrieve the newest element in the history
104     * @return the newest element
105     */
106    public double last(){
107   int idx = (this.l - 1)% H;
108   if(idx < 0){
109     idx = size()-1;
110   }
111   return this.q[idx];
112    }
113
114    /**
115     * Determine the number of elements in the history
116     * @return number of elements in the history
117     */
118    public int size(){
119   return this.s;
120    }
121
122    /**
123     * Determine whether the history is empty or not
124     * @return the size of the history
125     */
126    public boolean isEmpty(){
127   return this.size() == 0;
128    }
129
130    /**
131     * Append an element to the history and update the maximum value
132     * in amortized constant time
133     * @param v value to add in the history
134      */
135    public void add(double v){
136   /* remember the removed element */
137   this.old = first();
138   /* add the new value to the total sum */
139   this.sum += v;
140   this.s2 += (v*v);
141
142   /* v is the newest element in history */
143   q[l] = v;
144
145   /* update the maximum if needed */
146   if(v > this.max){
147     this.max = v;
148   }
149
150   /* move the "last" cursor by one to the right*/
151   l = (l+1)%H;
152
153   computeExpSum(v);
154
155   /* if  the size is exceeded */
156   if((this.size()) >= this.H){
157     /* mark as size exceed (must be marked only after the
158      * first time the size is exceeded to have a good
159      * variance estimate, see getVar()#double) */
160     this.exceed = true;
161
162     /* the removed element is not a part of the sum */
163     this.sum -= this.old;
164     this.s2 -= (this.old * this.old);
165
166     /* if the removed element was the maximum, check
167      * for the new maximum */
168     if(this.old == this.max && v < this.max){
169    this.max = v;
170    for(int i = 0; i < this.H; i++){
171      if(this.q[i] > this.max){
172     this.max = this.q[i];
173      }
174    }
175     }
176     /* move the "first" cursor by one to the right */
177     f = (f + 1)%H;
178   }
179   /* update the size */
180   this.s = Math.min(this.s +1,H);
181    }
182
183    /**
184     * @deprecated too complex
185     */
186    public double[] toArray(){
187   double t[] = new double[this.size()];
188   for(int i = 0; i < t.length; i++){
189     t[i] = this.q[(this.f+i)%this.H];
190   }
191   return t;
192    }
193
194    /**
195     * Retrieve the maximum value of the history
196     */
197    public double getMax(){
198   return this.max;
199    }
200
```

```
201     /**
202      * Computes an estimate of the variance of the elements of the history
203      * in constant time O(1)
204      *
205      * Let n = the number of measures
206      * S2 = \sum_i=1^n (x_i)^2
207      * var = \frac{S_2}{n} - avg^2
208      * URL: http://fr.wikipedia.org/wiki/Variance_%28statistiques%29
209      *
210      * NOTE: the exceed flag must be true only if an element has been
211      * removed from the history.
212      * @return  the variance of the elements in table supposing
213      *      equiprobability of elements
214      */
215     public double getVar(){
216    if(this.size() == 0){
217      return 0;
218    }
219
220    double avg = this.getAvg();
221    if((this.s2/this.size() - (avg*avg)) < 0){
222    }
223    return this.s2/this.size() - (avg*avg);
224     }
225
226     /**
227      * Computes the average of the elements of a table
228      * in constant time O(1)
229      * @return  average of the elements in table
230      */
231     public double getAvg(){
232    return this.sum/this.size();
233     }
234
235     /**
236      * Retrieve the exponential averaged sum of the elements in the
237      * history
238      * @return a(t) \cdot \gamma \cdot \left(\sum_{i=0}^h (1-\gamma)^i
239      *           \cdot \xi(A,B,t-i)\right)
240      */
241     public double getExpSum(){
242    return this.expSum;
243     }
244
245     /**
246      * Compute exponential averaged sum of the history
247      * in constant time O(1)
248      *
249      * NOTE 1:
250      * a(t) \cdot \gamma \cdot
251      *   \left(\sum_{i=0}^h (1-\gamma)^i \cdot \xi(A,B,t-i)\right)
252      * NOTE 2: the exceed flag must be true only if an element has been
253      * removed from the history. The value of the removed element is stored
254      * in old
255      * @param exp the new experience
256      */
257     private void computeExpSum(double exp){
258    /* get the core of the calculus (inside the parenthesis at
259     * line 2 in the note) */
260    this.expSum /= (this.age*Lib.gamma);
261
262    /* if one element has been removed, correct the sum */
263    if(this.exceed){
264      /* exceed implies the size is H */
265      this.expSum -= (this.gammaPowered * this.old);
266    }
267
268    /* increase the power of the average part in the sum */
269    this.expSum *= (1-Lib.gamma);
270
271    /* add the new experience: (1-\gamma)^0 \cdot \xi*/
272    this.expSum += exp;
273
274    /* compute the new value of the age factor */
275    this.age = Reputation.age(this.age, Lib.c_a);
276
277    /* multiply the sum by gamma and the age factor */
278    this.expSum = this.expSum * this.age * Lib.gamma;
279
280     }
281
282     /**
283      * Presents the elements of the history ordered by time
284      * @deprecated too complex
285      */
286     public String toString(){
287    double t[] = toArray();
288    String r = "";
289    for(int i = 0; i < t.length; i++)
290      r+= "\t"+t[i];
291
292    return r;
293     }
294   }
```

Listing B.8: `Lib.java`

```
1   /*-
2    * Copyright (c) 2006, 2007
3    *  Damien Saucez.  All rights reserved.
4    *
5    * Redistribution and use in source and binary forms, with or without
6    * modification, are permitted provided that the following conditions
7    * are met:
8    * 1. Redistributions of source code must retain the above copyright
9    *    notice, this list of conditions and the following disclaimer.
10   * 2. Redistributions in binary form must reproduce the above copyright
11   *    notice, this list of conditions and the following disclaimer in the
12   *    documentation and/or other materials provided with the distribution.
13   * 3. All advertising materials mentioning features or use of this software
14   *    must display the following acknowledgement:
15   *  This product includes software developed by Damien Saucez.
16   * 4. Neither the name of the University nor the names of its contributors
17   *    may be used to endorse or promote products derived from this software
18   *    without specific prior written permission.
19   *
20   * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
21   * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
22   * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
23   * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS
24   * BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
25   * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
26   * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
27   * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
28   * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
29   * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
30   * THE POSSIBILITY OF SUCH DAMAGE.
31   */
32  package simulator;
33  import java.util.Random;
34
35  /**
```

```
 * The class Lib proposes the implementation of usefull methods and is the
 * central point for the tuning parameters
 *
 * @author Damien Saucez
 * @version May 2007
 */
public class Lib{
  /**
   * Enumeration of the individual supported attacks
   * NONE: the node is not an attacker
   * CONSTANT: the node is a constant attacker
   * SAME: the node is a same attacker
   * REPULSION: the node is a repulsion attacker
   */
  public enum Attack {NONE, CONSTANT, RANDOM, SAME, REPULSION};
  /** Pseudo-random number generator */
  public static Random rand;
  /** The number of iterations after which the attack begins */
  public static int attackIter;
  /** Maximum number of iterations */
  public static int maxIter;
  /** Activate the reputation with true. Deactivate the
   * reputation with false */
  public static boolean reputationEnable;
  /** Size of the history */
  public static int H;// = 10;
  /** Value of \gamma */
  public static double gamma;// = 0.1;
  /** Value of c_a */
  public static double c_a;// = 0.01;
  /** Active to compute a reference instead of a NCS algorithm */
  public static boolean referenceRand = false;
  /** Prints the intermediate results on the standard output.
   * The intermediate result gives the sum of the relatives errors
   * of the correct node for each iteration. The output is presented
   * in 3 columns. The first is the iteration, the second the sum of
   * relatives errors and the thirds is the number of nodes used for
   * the sum. Theses results permit to compute the average relative
   * error for each iteration.
   * Activate this summary may considerabely increase the time of the
   * computation of coordinates due to its quadratic dependance with
   * the number of correct nodes in the system.
   */
  public static boolean intermediateResults = false;

  /**
   * Estimate the distance from A to B
   * @param   a   id of node A
   * @param   b   id of node B
   * @param   nc  nodes collector for A and B
   * @return  distance estimation ||\vec{x}_A - \vec{x}_B|| (\hat{d}_{AB})
   */
  public static double estimateDistance(int a, int b,
    final NodesCollector nc){
 Node na = nc.getNode(a);
 Node nb = nc.getNode(b);
 return na.getCoords(na).minus(nb.getCoords(nb)).norm();
  }

  /**
   * Compute local error between A and B. If A or B is/are attackers, the
   * error is 0 since it has no particular meaning in this case. The
   * relative error is also 0 if the real distance is unknown.
   *
   * @param   a   id of node A
   * @param   b   id of node B
   * @param   nc  nodes collector for A and B
   * @param   dc  distances collector for A and B
   * @return  local error estimator (\vec{x}_A -\vec{x}_b)^2
   */
  public static double localError(int a, int b,
      final NodesCollector nc, final DistancesCollector dc){
 Node na = nc.getNode(a);
 Node nb = nc.getNode(b);
 /* if one or both nodes are attackers, the relative error is 0 */
 if(na.attack != Lib.Attack.NONE || nb.attack != Lib.Attack.NONE){
   return 0;
 }
 double d_ab = estimateDistance(a, b, nc); /* the predicted distance */
 double _d_ab = dc.getDistance(a,b);     /* the measured distance */
 /* if the measured distance is unknown, the local error is 0 */
 if(_d_ab == -1){
   return 0;
 }
 return Math.pow(d_ab - _d_ab, 2);
  }

  /**
   * Return the relative error between a and b. If the real distance is
   * unknown, a -1 is returned.
   *
   * @param a id of node A
   * @param b i of node B
   * @param nc nodes collector for A and B
   * @param dc distances collector for A and B
   * @return relative error between (|d_{ab} - \hat{d}_{ab}|)/d_{ab}.
   *     If d_{ab} is unknown, a -1 is returned.
   */
  public static double relativeError(int a, int b,
    final NodesCollector nc, final DistancesCollector dc){
 double d_ab = estimateDistance(a, b, nc); /* the predicted distance */
 double _d_ab = dc.getDistance(a,b);     /* the measured distance */
 /* if the measured distance is unknown, the relative error is -1 */
 if(_d_ab == -1){
   return -1;
 }
 return Math.abs((d_ab - _d_ab))/_d_ab;
  }

  /**
   * Retrieve the variance of an array of numbers (n-1 degree of freedom).
   * If no numbers are in the table, the variance is 0.
   *
   * @deprecated too complex
   * @param table the array containing the number to compute the variance
   * @param avg  the average of table
   * @return the variance of the set of number in table
   */
  public static double var(double table[], double avg){
 if(table.length <= 0){
   return 0;
 }
 double s = 0;
 for(int i = 0; i < table.length; i++){
   s+= (table[i] - avg)*(table[i] - avg);
 }
 return s/table.length;
  }
}
```

Listing B.9: `Node.java`

```
/*-
```

```
 2   * Copyright (c) 2006, 2007
 3   *  Damien Saucez.  All rights reserved.
 4   *
 5   * Redistribution and use in source and binary forms, with or without
 6   * modification, are permitted provided that the following conditions
 7   * are met:
 8   * 1. Redistributions of source code must retain the above copyright
 9   *    notice, this list of conditions and the following disclaimer.
10   * 2. Redistributions in binary form must reproduce the above copyright
11   *    notice, this list of conditions and the following disclaimer in the
12   *    documentation and/or other materials provided with the distribution.
13   * 3. All advertising materials mentioning features or use of this software
14   *    must display the following acknowledgement:
15   *  This product includes software developed by Damien Saucez.
16   * 4. Neither the name of the University nor the names of its contributors
17   *    may be used to endorse or promote products derived from this software
18   *    without specific prior written permission.
19   *
20   * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
21   * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
22   * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
23   * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS
24   * BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
25   * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
26   * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
27   * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
28   * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
29   * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
30   * THE POSSIBILITY OF SUCH DAMAGE.
31   */
32  package simulator;
33  import java.util.List;
34  import java.util.ArrayList;
35  import java.util.Random;
36  import java.util.Set;
37  import java.util.HashSet;
38
39  /**
40   * The class Node implements basics function for virtual node of coordinate
41   * systems supported by the simulator
42   *
43   * @author Damien Saucez
44   * @version May 2007
45   */
46  public abstract class Node{
47    /**
48     * Attack performed by the node
49     */
50    public Lib.Attack attack;
51    /**
52     * Node's coordinates
53     */
54    private Coord c;
55    /**
56     * Node's identifier
57     */
58    private int nid;
59    /**
60     * Reference to main process
61     */
62    public Simulator simu;
63    /**
64     * Pseudo-random number generator for random attacks
65     */
66    private Random rRand;
67
68    /**
69     * Initialize the node. If Lib.referenceRand is true, the coordinates are
70     * randomly chosen in [0,500] at the beginning. Otherwise, the
71     * coordinates are initially set to 0.
72     *
73     * @param nid  identifier of the node
74     * @param simu reference to main process
75     */
76    public Node(int nid, final Simulator simu){
77  this.simu = simu;
78  this.setId(nid);
79  this.attack = Lib.Attack.NONE;
80  /* a normal simulation */
81  if(!Lib.referenceRand){
82    this.c = new EuclideanCoord(new double[]{0d,0d,0d});
83  }
84  /* a reference simulation */
85  else{
86    this.c = new EuclideanCoord(new double[]{
87   Math.random()*500d,Math.random()*500d,Math.random()*500d});
88  }
89  this.rRand = new Random(nid); /* generate a pseudo-random generator */
90   }
91
92   /* ============================== Identifier ============================== */
93   /**
94    * Set id of the node
95    * @param   nid new id of the node
96    * @return  previous id of the node
97    */
98   public int setId(int nid){
99  int oldNid = this.nid;
100 this.nid = nid;
101
102 return oldNid;
103  }
104  /**
105   * Retrieve the identifier of the node
106   * @param src node asking for the id
107   * @return current identifier of the node
108   */
109  public int getId(final Node src){
110 return this.nid;
111  }
112
113  /* ============================== Distances ============================== */
114  /**
115   * Retrieve distance from node n to current node
116   * @param   n   reference to the node
117   * @return  distance from node n to current node
118   */
119  public double getDistance(final Node n){
120 return this.simu.dc.getDistance(n.getId(n), this.getId(this));
121  }
122
123  /* ============================== Coordinates ============================== */
124  /**
125   * Set coordinates of the node
126   * @param   c   new coordinates of the node
127   * @return  previous coordinates of the node
128   */
129  public Coord setCoords(final Coord c){
130 Coord oldC = this.c;
131 this.c = c;
132 return oldC;
133  }
134
135  /**
```

```
136      * Retrieve the coordinates of the node. The resulting coordinates are
137      * the result of the attack process if the current node is an attacker.
138      * If src == this, the coordinates are the actual coordinates of the
139      * node.
140      *
141      * @param src node asking for the coordinates
142      * @return the coordinates of the node (could be fake coordinates)
143      */
144     public Coord getCoords(final Node src){
145    /* do not perform attack if the asking node is the current
146     * node itself */
147    if(src == this){
148      return this.c;
149    }
150
151    /* the result depends on the attack */
152    switch(this.attack){
153      case CONSTANT:
154     return getConstantCoords(src);
155      case RANDOM:
156     return getRandomCoords(src);
157      case SAME:
158     return getSameCoords(src);
159      case REPULSION:
160     return getRepulsionCoords(src);
161      default:
162     return getDefaultCoords(src);
163    }
164     }
165
166     /**
167      * Retrieve coordinates without any sort of attack process
168      * @param src node asking for the coordinates
169      * @return the coordinates with any modification
170      */
171     protected Coord getDefaultCoords(final Node src){
172    return this.c;
173     }
174
175     /**
176      * Retrieve coordinates with a constant attack
177      * @param src node asking for the coordinates
178      * @return the coordinates computed for a constant attack
179      */
180     protected Coord getConstantCoords(final Node src){
181    return this.c;
182     }
183
184     /**
185      * Retrieve coordinates with a random attack
186      * Random coordinates are randomly chosen in [0;500]
187      * @param src node asking for the coordinates
188      * @return the coordinates computed for a random attack
189      */
190     protected Coord getRandomCoords(final Node src){
191    Coord c = new EuclideanCoord(new double[]{
192      this.rRand.nextDouble()*500d,
193       this.rRand.nextDouble()*500d,
194       this.rRand.nextDouble()*500d});
195    return c;
196     }
197
198     /**
199      * Retrieve coordinates with a same attack
200      * @param src node asking for the coordinates
201      * @return the coordinates computed for a same attack
202      */
203     protected Coord getSameCoords(final Node src){
204    return src.getCoords(src);
205     }
206
207     /**
208      * Retrieve coordinates with a repulse attack
209      * @param src node asking for the coordinates
210      * @return the coordinates computed for a repulse attack
211      */
212     protected abstract Coord getRepulsionCoords(final Node src);
213
214     /* ============================ Processing ============================ */
215     /**
216      * Compute coordinates of node
217      */
218     public void computeCoords(){
219    /* if malicious node don't compute coordinates */
220    if(this.attack != Lib.Attack.NONE){
221      return;
222    }
223
224    Node n = getNextProcessNode();
225    double d = n.getDistance(this);
226
227    if(!preProcess(n,d))
228      return;
229    if(!process(n,d))
230      return;
231    if(!postProcess(n,d))
232      return;
233     }
234
235     /**
236      * Retrieve the reference of a surveyor of the node. After numerous
237      * calls, all the surveyors of the current node have been retrieved the
238      * same number of time.
239      * @return the identifier of a surveyor
240      */
241     public abstract Node getNextProcessNode();
242
243     /**
244      * Tasks to perform before computing new coordinates based on n
245      * @param n node used to compute new coordinates
246      * @param d distance from current node to node n
247      * @return True on success. Otherwise, a false is returned
248      */
249     protected abstract boolean preProcess(Node n, double d);
250
251     /**
252      * Compute coordinates of the current node based on node n (neighbour)
253      * @param n node used to compute new coordinates
254      * @param d distance from current node to node n
255      * @return True on success. Otherwise, a false is returned
256      */
257     protected abstract boolean process(Node n, double d);
258
259     /**
260      * Tasks to perform after computing new coordinates
261      * @param d distance from current node to node n
262      * @return True on success. Otherwise, a false is returned
263      */
264     protected abstract boolean postProcess(Node n, double d);
265
266     /**
267      * Gives a representation of the identifier as a string
268      * @return a string representation of the identifier of the node
269      */
```

```
270     public String toString(){
271   return ""+this.getId(this);
272     }
273 }
```

Listing B.10: NodesCollector.java

```
1  /*-
2   * Copyright (c) 2006, 2007
3   *  Damien Saucez.  All rights reserved.
4   *
5   * Redistribution and use in source and binary forms, with or without
6   * modification, are permitted provided that the following conditions
7   * are met:
8   * 1. Redistributions of source code must retain the above copyright
9   *    notice, this list of conditions and the following disclaimer.
10  * 2. Redistributions in binary form must reproduce the above copyright
11  *    notice, this list of conditions and the following disclaimer in the
12  *    documentation and/or other materials provided with the distribution.
13  * 3. All advertising materials mentioning features or use of this software
14  *    must display the following acknowledgement:
15  *  This product includes software developed by Damien Saucez.
16  * 4. Neither the name of the University nor the names of its contributors
17  *    may be used to endorse or promote products derived from this software
18  *    without specific prior written permission.
19  *
20  * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
21  * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
22  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
23  * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS
24  * BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
25  * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
26  * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
27  * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
28  * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
29  * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
30  * THE POSSIBILITY OF SUCH DAMAGE.
31  */
32 package simulator;
33 import java.util.Hashtable;
34 import java.util.Iterator;
35 import java.util.Enumeration;
36
37 /**
38  * Implements a collection of nodes with facilities to add nodes, get a
39  * specific node or list all the nodes.
40  *
41  * @author Damien Saucez
42  * @version May 2007
43  */
44 public class NodesCollector implements Iterable<Node>{
45   /** The collection of nodes */
46   private Hashtable<Integer,Node> nodes;
47
48   /**
49    * Instantiate a nodes collector
50    */
51   public NodesCollector(){
52  this.nodes = new Hashtable<Integer,Node>();
53   }
54
55   /**
56    * Retrieve node A
57    * @param a node id of A
58    * @return reference to node A
59    */
60   public Node getNode(int a){
61  return this.nodes.get(a);
62   }
63
64   /**
65    * Store a reference to node A
66    * @param n new reference of A
67    * @return previous reference to node A
68    */
69   public Node setNode(Node n){
70  int id = n.getId(n);
71  return this.nodes.put(n.getId(n), n);
72   }
73
74   /**
75    * List all the nodes in the collection of nodes
76    * @return an Iterator on the nodes of the collection
77    */
78   public Iterator<Node> iterator(){
79  return new EnumerationToIterator(nodes.elements());
80   }
81
82   /**
83    * Helper class to implement an iterator on the collection
84    * @see Iterator
85    */
86   private class EnumerationToIterator implements Iterator<Node>{
87  Enumeration<Node> e;
88  public EnumerationToIterator(Enumeration<Node> e){
89    this.e = e;
90  }
91  public boolean hasNext(){
92    return e.hasMoreElements();
93  }
94  public Node next(){
95    return e.nextElement();
96  }
97  public void remove(){
98    throw new UnsupportedOperationException();
99  }
100
101  }
102 }
```

Listing B.11: Opinion.java

```
1  /*-
2   * Copyright (c) 2006, 2007
3   *  Damien Saucez.  All rights reserved.
4   *
5   * Redistribution and use in source and binary forms, with or without
6   * modification, are permitted provided that the following conditions
7   * are met:
8   * 1. Redistributions of source code must retain the above copyright
9   *    notice, this list of conditions and the following disclaimer.
10  * 2. Redistributions in binary form must reproduce the above copyright
11  *    notice, this list of conditions and the following disclaimer in the
12  *    documentation and/or other materials provided with the distribution.
13  * 3. All advertising materials mentioning features or use of this software
14  *    must display the following acknowledgement:
15  *  This product includes software developed by Damien Saucez.
16  * 4. Neither the name of the University nor the names of its contributors
17  *    may be used to endorse or promote products derived from this software
18  *    without specific prior written permission.
```

```
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS
 * BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
 * THE POSSIBILITY OF SUCH DAMAGE.
 */
package simulator;

/**
 * The class Opinion implements the Opinion space presented in the thesis
 * and all the operations on it. For more information, please refer to the
 * master thesis document.
 *
 * @author Damien Saucez
 * @version May 2007
 */
public class Opinion{
  /** Belief */
  public double b;
  /** Disbelief */
  public double d;
  /** Uncertainty */
  public double u;

  /**
   * Instantiate the opinion with the knowledge of the opinion
   * components
   * @param b belief
   * @param d disbelief
   * @param u uncertainity
   */
  public Opinion(double b, double d, double u){
 this.b = b;
 this.d = d;
 this.u = u;
  }

  /**
   * Compute the discounting of current opition with the option of right
   * @param right the right term of the discounting operator
   * @return this (x) right
   */
  public Opinion discounting(final Opinion right){
 double b = this.b * right.b;
 double d = this.b * right.d;
 double u = this.d + this.u + this.b * right.u;

 return new Opinion(b,d,u);
  }

  /**
   * Compute the consensus of current opinion with the opinion of right
   * @param right the right term of the consensus operator
   * @return this (+) B
   */
  public Opinion consensus(final Opinion right){
 double kappa = this.u + right.u - this.u * right.u;
 double b = (this.b*right.u + right.b*this.u)/kappa;
 double d = (this.d*right.u + right.d*this.u)/kappa;
 double u = (this.u * right.u)/kappa;

 return new Opinion(b,d,u);
  }

  /**
   * Compute the scalar value of the opinion
   * @return a scalar representation of the opinion tuple
   */
  public double toTrust(){
 return this.b * (1-this.u);
  }
}
```

Listing B.12: RCA.java

```
/*-
 * Copyright (c) 2006, 2007
 *  Damien Saucez.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *  This product includes software developed by Damien Saucez.
 * 4. Neither the name of the University nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS
 * BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
 * THE POSSIBILITY OF SUCH DAMAGE.
 */
package simulator;
import java.util.Set;
import java.util.Stack;

/**
 * Implements a Reputation Computation Agent (RCA). The RCA is responsible
 * to compute the reputation of nodes and gives tickets to all the nodes.
 *
 * @author Damien Saucez
 * @version May 2007
 */
public class RCA{
  /**
   * Reference to main process
   */
  private Simulator simu;
  /**
   * Variation history vector for surveyors
   */
```

```
 private History[] vhvs;
 /**
  * Trust to the surveyors
  */
 private Trust[] trusts;

 /**
  * Instanciate the RCA of the simulator simu
  * @param simu the simulator responsible of the RCA
  */
 public RCA(final Simulator simu){
this.simu = simu;
this.vhvs = new History[simu.nbNodes];
this.trusts = new Trust[simu.nbNodes];

for(int i = 0; i < simu.nbNodes; i++){
  this.vhvs[i] = new History(Lib.H);
  this.vhvs[i].add(0);
  this.trusts[i] = new Trust();
}
 }

 /**
  * Validate the coordinates of a node.
  * If the reputation is enabled, the coordinates of the node are updated
  * and its coordinates are set by the RCA. If the reputation is disabled,
  * the reputation is set to 1 and the coordinates are updated.
  * @param rn node to validate
  * @param newC new coordinates of the node
  */
 public void validate(ReputationNode rn, final Coord newC){
/**
 * If the reputation is not enabled, do not compute reputation
 */
if(!Lib.reputationEnable){
  rn.setReputation(1);
  ((Node)rn).setCoords(newC);
  return;
}

double reput = 0;
int nid = ((Node)rn).getId((Node)rn);

/* diff = || \vec{c}_t - \vec{c}_{t-1} || */
double diff = newC.minus(((Node)rn).getCoords((Node)rn)).norm();
/* add the variation to the history */

this.vhvs[nid].add(diff);

/* Get all the surveyors */
Set<Surveyor> survs = rn.getSurveyors();

double n = computeN(survs);

/* for each surveyor */
/* create the stack that store the temporar results */
Stack<Opinion> st = new Stack<Opinion>();
double expRCA; /* experience view with the RCA */
Opinion wRCAS; /* opinion from the RCA to the surveyor */
Opinion wSA; /* opinion from the surveyor and the node */
Opinion w;  /* discounting of the results */
for(Surveyor s : rn.getSurveyors()){
  /* compute trust RCA->s */
  wRCAS = reputationRCA2S(s, (Node)rn, n);
  /* compute trust s->rn */
  wSA = reputationS2A(s, (Node)rn, n);
  /* combine trusts RCA->s(x) s->rn */
  w = wRCAS.discounting(wSA);
  /* push the result on the computation stack */
  st.push(w);
}

/* combine all the results (+) RCA->s (x) s->rn */
Opinion oa;
Opinion ob;
while(st.size() >= 2){
  oa = st.pop();
  ob = st.pop();
  st.push(oa.consensus(ob));
}

/* Convert the opinion to its scalar value */
if(st.size() == 1){
  reput = st.pop().toTrust();
}

/* Give the validation to the node and its coordinates */
double oRep = rn.setReputation(reput);
((Node)rn).setCoords(newC);
 }

 /**
  * Compute the opinion RCA has in s, the surveyor of the node to validate
  * @param s   the surveyor node
  * @param a   the surveyed node
  * @param n   the normalization factor (generated with computeN)
  * @return \tilde{\omega}^{RCA}_{s}
  */
 private Opinion reputationRCA2S(Surveyor s, Node a, double n){
/* identifier of the surveyor */
int sid  = ((Node)s).getId((Node)s);

/* get information */
Trust t = this.trusts[sid];

/* compute the experience and return the opinion */
double exp = Reputation.experienceRCA(this.vhvs[sid], n);
return t.updateTrust(exp);
 }

 /**
  * Compute the opinion s has in a
  * @param s   the surveyor node
  * @param a   the surveyed node
  * @param n the normalization factor (generated with computeN)
  * @return   \omega^{s}_{a}
  */
 private Opinion reputationS2A(Surveyor s, Node a, double n){
return s.computeTrust(a);
 }


 /**
  * Computes n = argmax(\cup_{H_n \in S_A} v_A^{H_n}), the normalization
  * factor
  * @param   survs all the surveyors of the node
  * @return  the normalization factor
  */
 public double computeN(final Set<Surveyor> survs){
double n = -1;

/* déterminer n:*/
for(Surveyor s : survs){
  /* récupérer la valeur maximale du VHV de chaque surveillant */
```

```
    double max = this.vhvs[((Node)s).getId((Node)s)].getMax();

    /* set max to the maximum value between max and n*/
    if(max > n){
  n = max;
    }
  }
  return n;
   }
}
```

Listing B.13: RVivaldiNode.java

```
/*-
 * Copyright (c) 2006, 2007
 *  Damien Saucez.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *  This product includes software developed by Damien Saucez.
 * 4. Neither the name of the University nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS
 * BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
 * THE POSSIBILITY OF SUCH DAMAGE.
 */
package simulator;
import java.util.Set;
import java.util.HashSet;

/**
 * Implement a node performing RVivaldi
 *
 * @author Damien Saucez
 * @version May 2007
 * @see Node
 * @see ReputationNode
 */
public class RVivaldiNode extends VivaldiNode
          implements ReputationNode,Surveyor{
  /**
   * Reputation
   */
  private double reput;
  /**
   * Set of surveyors
   */
  protected Set<Surveyor> surveyors;

  /**
   * Instantiate a node respecting RVivaldi paradigm
   * The initial reputation is set to 0.5
   *
   * @param nid identifier of the node
   * @param simu reference to main process
   */
  public RVivaldiNode(int nid, final Simulator simu){
super(nid, simu);
this.surveyors = new HashSet<Surveyor>();
this.setReputation(.5);
  }

  /* ============================== Inherited ============================== */
  protected boolean process(Node n, double d){
/* computes new coordinates and local error with RVivaldi */
RVivaldiNode rvn = (RVivaldiNode)n;
CE_r nc = this.rvivaldi(d,
 this.getCoords(this),
 rvn.getCoords(this),
 this.e_i,
 rvn.getLocalError(this),
 ((ReputationNode)n).getReputation());
this.e_i = nc.e_i;

/* validate the new coordinates */
this.simu.rca.validate((ReputationNode)this, nc.coord);
return true;
  }

  /* ============================== Reputation ============================== */
  public double getReputation(){
if(!Lib.reputationEnable){
  return 1;
}
return this.reput;
  }

  public double setReputation(double r){
double oldR = this.reput;
this.reput = r;
return oldR;
  }

  /* ============================== Surveyors ============================== */
  /**
   * Add surveyor s to the set of surveyors of the current node
   * @param s the surveyor to add
   */
  public void addSurveyor(final Surveyor s){
this.surveyors.add(s);
  }

  /**
   * Get all the surveyors of the current node
   * @return the set of all the surveyors of the node
   */
  public Set<Surveyor> getSurveyors(){
return this.surveyors;
  }

  /* ============================== Attacks ============================== */
  /**
   * @see Node
   * NOTE: the attacker validates its fake coordinates
```

```
120      */
121     protected Coord getConstantCoords(final Node src){
122    Coord c = super.getConstantCoords(src);
123    this.simu.rca.validate((ReputationNode)this, c);
124    return c;
125     }
126
127     /**
128      * @see Node
129      * NOTE: the attacker validates its fake coordinates
130      */
131     protected Coord getRandomCoords(final Node src){
132    Coord c = super.getRandomCoords(src);
133    this.simu.rca.validate((ReputationNode)this, c);
134    return c;
135     }
136
137     /**
138      * @see Node
139      * NOTE: the attacker validates its fake coordinates
140      */
141     protected Coord getSameCoords(final Node src){
142    Coord c = super.getSameCoords(src);
143    this.simu.rca.validate((ReputationNode)this, c);
144    return c;
145     }
146
147     /**
148      * @see Node
149      * NOTE: the attacker validates its fake coordinates
150      */
151     protected Coord getRepulsionCoords(final Node src){
152    Coord c = super.getRepulsionCoords(src);
153    this.simu.rca.validate((ReputationNode)this, c);
154    return c;
155     }
156
157     /* ========================== RVivaldi ========================== */
158     /**
159      * Computes new coordinates and new local error of i respecting RVivaldi
160      * algorithm based on observations of j
161      * @param rtt distance from i to j
162      * @param x_i coordinates of i
163      * @param x_j coordinates of j
164      * @param e_i local error of i
165      * @param e_j local error of j
166      * @param reput reputation of j
167      * @return the tuple containing the new coordinates of i and its new
168      *   local error
169      */
170     protected CE_r rvivaldi(double rtt, Coord x_i, final Coord x_j,
171       double e_i, double e_j, double reput){
172    /* résultat (coords,e_i) */
173    CE_r result = new CE_r();
174
175    /* x_i - x_j */
176    Coord x_i_x_j = x_i.minus(x_j);
177    /* | x_i - x_j || */
178    double norm_x_i_x_j = x_i_x_j.norm();
179
180    /* (1) */
181    double w = e_i/(e_i+e_j);
182
183    /* (2) */
184    double e_s =  Math.abs(norm_x_i_x_j - rtt)/rtt;
185
186    /* (3) */
187    result.e_i = e_s * this.c_e * w + e_i * (1 - this.c_e * w);
188
189    /* (4) */
190    double delta = this.c_c * w;
191
192    /* u(x_i - x_j) */
193    Coord dir = x_i_x_j.toUnitary();
194
195    /* the change is here */
196    /* reput * delta * (rtt - || x_i - x_j||) */
197    double tmp1 = reput * delta * (rtt - norm_x_i_x_j);
198    /* reput * delta * (rtt - || x_i - x_j||) * u(x_i - x_j) */
199    Coord tmp2 = dir.leftDotProduct(tmp1);
200
201    /* xi = x_i + reput * delta * (rtt - || x_i - x_j||) * u(x_i - x_j) */
202    result.coord = x_i.plus(tmp2);
203
204    return result;
205     }
206   }
```

Listing B.14: `Reputation.java`

```
1   /*-
2    * Copyright (c) 2006, 2007
3    *  Damien Saucez.  All rights reserved.
4    *
5    * Redistribution and use in source and binary forms, with or without
6    * modification, are permitted provided that the following conditions
7    * are met:
8    * 1. Redistributions of source code must retain the above copyright
9    *    notice, this list of conditions and the following disclaimer.
10   * 2. Redistributions in binary form must reproduce the above copyright
11   *    notice, this list of conditions and the following disclaimer in the
12   *    documentation and/or other materials provided with the distribution.
13   * 3. All advertising materials mentioning features or use of this software
14   *    must display the following acknowledgement:
15   *  This product includes software developed by Damien Saucez.
16   * 4. Neither the name of the University nor the names of its contributors
17   *    may be used to endorse or promote products derived from this software
18   *    without specific prior written permission.
19   *
20   * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
21   * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
22   * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
23   * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS
24   * BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
25   * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
26   * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
27   * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
28   * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
29   * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
30   * THE POSSIBILITY OF SUCH DAMAGE.
31   */
32  package simulator;
33
34  /**
35   * The class Reputation gives usefull methods to work with the reputation
36   * concept
37   *
38   * @author Damien Saucez
39   * @version May 2007
40   */
41  public class Reputation{
42    /**
```

```
    * Compute experience A has in B using the formula:
    *  1 - \frac{|\hat{d}_{AB} - d_{AB}|}{max(d_{AB},\hat{d}_{AB})}
    *
    * @param a id of node A
    * @param b id of node B
    * @param dc distances collector for A and B
    * @param nc nodes collector for A and B
    * @return experience node A has in node B
    */
   public static double experience(int a, int b, DistancesCollector dc,
     NodesCollector nc){
  double estimated = Lib.estimateDistance(a, b, nc);
  double measured = dc.getDistance(a, b);
  if(measured == -1){
    System.err.println("DISTANCE - 1 exp");
  }

  return 1-Math.abs(estimated - measured)/Math.max(estimated, measured);
   }

   /**
    * Computes the new aging factor based on the previous
    * @param age the previous ageing factor
    * @param c_a the age bonus coefficient to use
    * @return the new value of the ageing factor
    */
   public static double age(double age, double c_a){
  return c_a + (1-c_a)*age;
   }

   /**
    * Compute the experience as produced by the RCA. An experience of 0.5 is
    * given if the RCA is not able to compute the experience.
    *
    * @param vhv the variation history vector for the node experimented
    * @param n the normalization factor
    * @return the experience the RCA with the node with the variation
    *     history vector vhv. On error, return 0.5.
    */
   public static double experienceRCA(final History vhv, double n){
  if(n==0 || vhv.size() == 0){
    return 0.5;
  }
  try{
    return 1.0 - Math.sqrt(vhv.getVar())/(n*vhv.size());
  }catch(NumberFormatException e){return 0.5;}
   }

   /**
    * Compute the trust based on the history of the experiences
    * @param gamma weight factor
    * @param age age factor
    * @param h history vector
    * @return the trust infered from the history
    */
   public static Opinion trust(double gamma, double age, final History h){
  /* Compute the belief with the exponential sum */
  double b = h.getExpSum();
  /* Compute the disbelief */
  double d = 1 - b;
  /* Get the variance of the history*/
  double u = h.getVar();

  /* to respect the additive function theorem */
  b = b/(b+d+u);
  d = d/(b+d+u);
  u = u/(b+d+u);

  return new Opinion(b, d, u);
   }
}
```

Listing B.15: ReputationNode.java

```
/*-
 * Copyright (c) 2006, 2007
 *  Damien Saucez.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *  This product includes software developed by Damien Saucez.
 * 4. Neither the name of the University nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS
 * BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
 * THE POSSIBILITY OF SUCH DAMAGE.
 */
package simulator;
import java.util.Set;

/**
 * Gives methods for paradigms supporting the reputation model
 *
 * @author Damien Saucez
 * @version May 2007
 */
public interface ReputationNode{
  /**
   * Retrieve the node's reputation
   * @return current reputation of the node
   */
  public double getReputation();

  /**
   * Set reputation to r and return the old value
   * @param r new value of the reputation
   * @return old reputation value
   */
  public double setReputation(double r);

  /**
   * Return the refereces set to all the surveyors of the node
   * @return the surveyors
   */
```

```
59    public Set<Surveyor> getSurveyors();
60    /**
61     * Add the surveyor s to the node
62     * @param s the surveyor to add
63     */
64    public void addSurveyor(final Surveyor s);
65  }
```

Listing B.16: Simulator.java

```
1   /*-
2    * Copyright (c) 2006, 2007
3    *  Damien Saucez.  All rights reserved.
4    *
5    * Redistribution and use in source and binary forms, with or without
6    * modification, are permitted provided that the following conditions
7    * are met:
8    * 1. Redistributions of source code must retain the above copyright
9    *    notice, this list of conditions and the following disclaimer.
10   * 2. Redistributions in binary form must reproduce the above copyright
11   *    notice, this list of conditions and the following disclaimer in the
12   *    documentation and/or other materials provided with the distribution.
13   * 3. All advertising materials mentioning features or use of this software
14   *    must display the following acknowledgement:
15   *  This product includes software developed by Damien Saucez.
16   * 4. Neither the name of the University nor the names of its contributors
17   *    may be used to endorse or promote products derived from this software
18   *    without specific prior written permission.
19   *
20   * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
21   * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
22   * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
23   * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS
24   * BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
25   * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
26   * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
27   * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
28   * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
29   * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
30   * THE POSSIBILITY OF SUCH DAMAGE.
31   */
32  package simulator;
33  import java.util.Random;
34
35  /**
36   * @author Damien Saucez
37   * @version May 2007
38   */
39  public class Simulator{
40    /**
41     * Distances collector
42     */
43    public DistancesCollector dc;
44    /**
45     * Nodes collector
46     */
47    public NodesCollector nc;
48    /**
49     * Number of nodes in the system
50     */
51    public int nbNodes;
52
53    public RCA rca;
54
55    public static void main(String argv[]){
56    Lib.attackIter = Integer.parseInt(argv[1]);
57    Lib.maxIter = Integer.parseInt(argv[2]);
58    Lib.reputationEnable = Boolean.parseBoolean(argv[3]);
59    Lib.H = Integer.parseInt(argv[4]);
60    Lib.gamma = Double.parseDouble(argv[5]);
61    Lib.c_a = Double.parseDouble(argv[6]);
62    int nbNodes = Integer.parseInt(argv[7]);
63    Lib.rand = new Random(nbNodes);
64
65    Simulator s = new Simulator(nbNodes, argv[0]);
66    s.run();
67     }
68
69     /**
70      * Initialize the simulator
71      * @param nbNodes number of nodes in the system
72      * @param attackersFileName file containing the specification of
73      *     attackers
74      */
75     public Simulator(int nbNodes, final String attackersFileName){
76    this.nbNodes = nbNodes;
77    /* initialiser le Node Collector */
78    this.nc = new NodesCollector();
79    /* initialiser le Distances Collector */
80    this.dc = new FlatDistancesCollector(this.nbNodes);
81
82    /* create all the nodes */
83    for(int nid = 0; nid < nbNodes; nid++){
84      Node n = new RVivaldiNode(nid, this);
85      nc.setNode(n);
86    }
87
88    /* set the neighbours to all nodes */
89    Configurator.configNeighbours(this, "config/flat_neighs");
90    /* Load the RTT matrix */
91    Configurator.configDistances(this, "config/flat_rtts");
92    /* Set the type of attack for all the nodes */
93    Configurator.configAttackants(this, attackersFileName);
94    /* Set the surveyors */
95    Configurator.configSurveyrors(this, "config/survs");
96    this.rca = new RCA(this);
97     }
98
99     public void run(){
100   System.out.println("# AttackIter: " + Lib.attackIter);
101   System.out.println("# MaxIter: " + Lib.maxIter);
102   System.out.println("# Reputation?: "+((Lib.reputationEnable)?"1":"0"));
103   System.out.println("# H: " + Lib.H);
104   System.out.println("# Gamma: " + Lib.gamma);
105   System.out.println("# c_a: " + Lib.c_a);
106
107   /* Compute coordinates */
108   int l = 0;
109   double sum = 0;
110   int cnt = 0;
111   double re;
112
113   /* If the simulator must simulate a NCS */
114   if(!Lib.referenceRand){
115     while(l++ < Lib.maxIter){
116    /* computes the coordinates for each node */
117    for(Node n : nc){
118      n.computeCoords();
119    }
120
121    /* Print a summary after each iteration if asked
122     * Attention: This computation may take long time
```

```
     */
    if(Lib.intermediateResults){
      for(int i = 0; i < nbNodes; i++){
     for(int j = i; j < nbNodes; j++){
       if(nc.getNode(i).attack != Lib.Attack.NONE ||
          nc.getNode(j).attack != Lib.Attack.NONE){
      continue;
       }
       re = Lib.relativeError(i, j, nc, dc);
       if(re != -1 && i != j){
      sum += re;
      cnt++;
       }
     }
      }
      System.out.println(l+"\t" + sum + "\t" + cnt);
      sum = 0;
      cnt = 0;
    }
     }
   }

   /* Print the relative error between any two correct nodes. The
    * output has 3 columns. First, the id of the source of estimation.
    * Second, the destination of the estimation. Third, the relative
    * error between node from first column and second column.
    */
   for(int i = 0; i < nbNodes; i++){
     for(int j = i; j < nbNodes; j++){
    if(nc.getNode(i).attack != Lib.Attack.NONE ||
       nc.getNode(j).attack != Lib.Attack.NONE){
      continue;
    }
    re = Lib.relativeError(i, j, nc, dc);
    if(re != -1 && i != j){
      System.out.println(i+"\t"+j+"\t"+re);
    }
     }
   }


   }
}
```

Listing B.17: `Surveyor.java`

```
/*-
 * Copyright (c) 2006, 2007
 *  Damien Saucez.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *  This product includes software developed by Damien Saucez.
 * 4. Neither the name of the University nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS
 * BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
 * THE POSSIBILITY OF SUCH DAMAGE.
 */
package simulator;

/**
 * Gives methods for nodes able to become surveyors
 *
 * @author Damien Saucez
 * @version May 2007
 */
public interface Surveyor{
  /**
   * Compute the trust the surveyor has in a
   */
  public Opinion computeTrust(Node a);
}
```

Listing B.18: `SurveyorNode.java`

```
/*-
 * Copyright (c) 2006, 2007
 *  Damien Saucez.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *  This product includes software developed by Damien Saucez.
 * 4. Neither the name of the University nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS
 * BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
 * THE POSSIBILITY OF SUCH DAMAGE.
 */
package simulator;
import java.util.Hashtable;

/**
 * Implementation of a node able to survey other nodes
```

```
 *
 * @author Damien Saucez
 * @version May 2007
 */
public abstract class SurveyorNode extends Node implements Surveyor{
  /**
   * Collection of trust to nodes
   */
  private Hashtable<Integer,Trust> trusted;

  /**
   * Instanciate a node which can be a surveyor
   * @param nid identifier of the node
   * @param simu reference to main process
   */
  public SurveyorNode(int nid, final Simulator simu){
 super(nid, simu);
 this.trusted = new Hashtable<Integer,Trust>();
  }

  /**
   * Compute the trust to node n
   * @param n the node to trust
   * @return the opinion from this node to n
   */
  public Opinion computeTrust(Node n){
 /* compute experience this->n */
 double exp = Reputation.experience(this.getId(this), n.getId(n),
  this.simu.dc, this.simu.nc);

 /* retrieve the trust for n */
 Trust t = this.trusted.get(n.getId(n));
 /* add to the list if never trusted before */
 if(t == null){
   t = new Trust();
   this.trusted.put(n.getId(n), t);
 }
 /* compute the opinion with the experience */
 return t.updateTrust(exp);
  }

}
```

Listing B.19: `Trust.java`

```
/*-
 * Copyright (c) 2006, 2007
 *  Damien Saucez.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *  This product includes software developed by Damien Saucez.
 * 4. Neither the name of the University nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS
 * BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
 * THE POSSIBILITY OF SUCH DAMAGE.
 */
package simulator;

/**
 * Abstraction to compute the trust based on the last Lib.H experiences
 *
 * @author Damien Saucez
 * @version May 2007
 */
public class Trust{
  /**
   * Ageing factor
   */
  private double age;
  /**
   * vector of experiences
   */
  private History experiences;

  /**
   * Instanciate the capability to compute the trust for a particular
   * node based on the last Lib.H experiences
   */
  public Trust(){
 this.experiences = new History(Lib.H);
 this.age = 0;
  }

  /**
   * Compute the new trust with the new experience exp
   * @param exp the last experience for the node
   * @return the opinion based on the experiences
   */
  public Opinion updateTrust(double exp){
 /* add the experience to the history */
 this.experiences.add(exp);

 /* compute the new age factor */
 this.age = Reputation.age(this.age, Lib.c_a);

 /* compute the new trust and return the opinion */
 return Reputation.trust(Lib.gamma, this.age, this.experiences);
  }

}
```

Listing B.20: `VivaldiNode.java`

```
/*-
 * Copyright (c) 2006, 2007
 *  Damien Saucez.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
```

```
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *  This product includes software developed by Damien Saucez.
 * 4. Neither the name of the University nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS
 * BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
 * THE POSSIBILITY OF SUCH DAMAGE.
 */
package simulator;
import java.util.List;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.StringTokenizer;
import java.util.Hashtable;
import java.util.Random;

/**
 * Implements nodes with Vivaldi paradigm able to survey other nodes but
 * without the support of the reputation
 *
 * @author Damien Saucez
 * @version May 2007
 */
public class VivaldiNode extends SurveyorNode implements Surveyor{
  /**
   * References set to neighbours
   */
  protected List<Node> neighs;
  /**
   * Counter used to move inside neighs
   */
  private int countN;
  /**
   * Coordinates of the target for repulse attacks
   */
  public Coord target;
  /** local error */
  public double e_i;
  /** error c_e coeficient factor */
  protected double c_e = 0.25;
  /** delta c_c coeficient factor */
  protected double c_c = 0.25;

  /**
   * Instantiate a node respecting Vivaldi paradigm
   * @param nid identifier of the node
   * @param simu reference to main process
   */
  public VivaldiNode(int nid, final Simulator simu){
  super(nid, simu);
  this.e_i = 1;
  this.neighs = new ArrayList<Node>();
  this.countN = 0;
  }

  /* ============================== Inherited ============================== */
  public Node getNextProcessNode(){
  countN = (countN+1)%this.neighs.size();
  return this.neighs.get(countN);
  }

  protected boolean preProcess(Node n, double d){
  return true;
  }

  protected boolean process(Node n, double d){
  VivaldiNode vn = (VivaldiNode)n;
  CE_r nc = this.vivaldi(d,
  this.getCoords(this),
  vn.getCoords(this),
  this.e_i,
  vn.getLocalError(this));
  setCoords(nc.coord);
  this.e_i = nc.e_i;
  return true;
  }

  protected boolean postProcess(Node n, double d){
  return true;
  }

  protected Coord getRepulsionCoords(final Node src){
  return getDefaultCoords(src);
  }

  public double getDistance(final Node n){
  Random rand = new Random(this.getId(this));

  if(this.attack != Lib.Attack.REPULSION){
    return super.getDistance(n);
  }
  Coord current = n.getCoords(n);
  double norm = target.minus(current).norm();

  double delta = this.c_c * ((VivaldiNode)n).getLocalError(n)/
    (((VivaldiNode)n).getLocalError(n) + this.getLocalError(this));

  return (norm/delta + norm);
  }

  /* ============================== Vivaldi ============================== */

  /**
   * Add a node to the neighbour list
   * @param   n   reference to the neighbour
   */
  public void addNeighbour(final Node n){
  this.neighs.add(n);
  }

  /**
   * Retrieve Vivaldi local error
   * @param src node asking for the local error
   * @return local error of current node
   */
  public double getLocalError(final Node src){
  return this.e_i;
  }
```

```
142
143     /**
144      * Computes new coordinates and new local error of i respecting Vivaldi
145      * algorithm based on observations of j
146      * @param rtt distance from i to j
147      * @param x_i coordinates of i
148      * @param x_j coordinates of j
149      * @param e_i local error of i
150      * @param e_j local error of j
151      * @return the tuple containing the new coordinates of i and its new
152      *     local error
153      */
154     protected CE_r vivaldi(double rtt, Coord x_i, final Coord x_j,
155       double e_i, double e_j){
156     // résultat (coords,e_i)
157     CE_r result = new CE_r();
158
159     // x_i - x_j
160     Coord x_i_x_j = x_i.minus(x_j);
161     //| x_i - x_j ||
162     double norm_x_i_x_j = x_i_x_j.norm();
163
164     // (1)
165     double w = e_i/(e_i+e_j);
166
167     // (2)
168     double e_s =  Math.abs(norm_x_i_x_j - rtt)/rtt;
169
170     // (3)
171     result.e_i = e_s * this.c_e * w + e_i * (1 - this.c_e * w);
172
173     // (4)
174     double delta = this.c_c * w;
175
176     // u(x_i - x_j)
177     Coord dir = x_i_x_j.toUnitary();
178
179     // delta * (rtt - || x_i - x_j||)
180     double tmp1 = delta * (rtt - norm_x_i_x_j);
181     // delta * (rtt - || x_i - x_j||) * u(x_i - x_j)
182     Coord tmp2 = dir.leftDotProduct(tmp1);
183
184     // xi = x_i + delta * (rtt - || x_i - x_j||) * u(x_i - x_j)
185     result.coord = x_i.plus(tmp2);
186
187     return result;
188
189     }
190
191     /**
192      * Helper class to simplify the representation of results of a Vivaldi
193      * run
194      */
195     protected class CE_r{
196     public Coord coord;
197     public double e_i;
198
199     public CE_r(){}
200     public CE_r(Coord coord, double e_i){
201       this.coord = coord;
202       this.e_i = e_i;
203     }
204     }
205 }
```