UNIVERSITÉ DE LIÈGE
Faculté des Sciences Appliquées
Institut d'Électricité Montéfiore
RUN - Research Unit in Networking

Improving Overlay Routing scalability using an Internet Coordinate System

François Cantin
Licencié en Informatique
DEA interuniversitaire en Informatique
Boursier FRIA

Submitted for the doctoral
Degree in Computer Science

April 2012

Abstract

Nowadays lots of real time applications are used over the Internet: voice over IP, online video games, *etc.* For these applications the performance of the path between two communicating nodes is critical. Particularly, most of these applications require small delays between communicating nodes. For these applications, the problem is that the choice of the routes in the Internet is generally not very much guided by performance concerns. It is well known that for lots of node pairs the default Internet path is suboptimal and there exists an alternative path providing a smaller delay between these nodes. In this thesis, we mainly address the problem of finding these alternative paths.

Replacing Internet's routing philosophy in order to obtain default paths providing the best performance possible should be a good theoretical solution. However, replacing Internet's routing philosophy by a brand new one is very difficult or even impossible in practice. Another solution is to leave the default routes as they are and to perform indirect routing. Consider a path AB between two nodes A and B. If a path ACB has a smaller delay than AB, then, instead of sending data directly to B, A can send them to C and C can relay them to B. This is called overlay routing because we manage the routing in an overlay network built on top of the Internet (*i.e.* at the application level). Overlay routing is a promising way to improve the quality of service in the Internet but its main drawback is its poor scalability: measuring the characteristics of the paths, exchanging the measurement results between the nodes and computing the best routes in the full mesh overlay network generally implies a high consumption of resources. In this thesis, our main contribution is the design of a lightweight one-hop overlay routing mechanism improving the latencies: we define criteria that rely on the information provided by an Internet Coordinate System (ICS) in order to provide a small set of potential one-hop shortcuts for any given path in the network with small costs. Our best criterion does not guarantee to find the best shortcut for any given path in a network but, even in networks with hundreds or thousands of nodes, it will restrict the search for potential shortcuts to about one or two percent of the total number of nodes.

Even if the estimation-based approach of overlay routing is our main contribution, this thesis also presents general results about routing shortcuts and Internet Coordinate Systems. For an ICS, a routing shortcut is a Triangle Inequality Violation (TIV) and it is often a big problem. Indeed, a TIV will cause estimation errors since, in this particular case, nodes cannot be embedded into any metric space. In this thesis, we study TIVs existing in the Internet and their impact on the Vivaldi ICS. This analysis leads to two contributions. Firstly, we propose criteria to establish, with a high probability of success, if there exists a shortcut or not for a given path AB. Secondly, we propose a Two-Tier architecture for ICSes that mitigates the effect of TIVs on the estimations. Finally, this thesis also discusses the efficiency of two solutions proposed in the literature in order to obtain an ICS that can deal with TIVs. The first one consists in applying non-linear transformations to delays before trying to embed them in a metric space. The second one consists in modelling the estimation problem as a matrix completion problem in order to completely avoid the embedding in a metric space.

Résumé

Aujourd'hui de nombreuses applications communiquant en temps réel sont déployées dans l'Internet : téléphonie IP, jeux vidéos en ligne, etc. Pour de telles applications, les performances d'un chemin entre deux nœuds sont un point critique. En particulier, la plupart de ces applications exigent que les délais entre les nœuds soient les plus petits possible. Le problème est que le choix des routes dans l'Internet n'est généralement pas basé sur les performances et il est bien connu que, pour de nombreux chemins dans l'Internet, il existe un chemin alternatif proposant un délai plus court. Dans cette thèse, notre objectif principal est de trouver ces chemins alternatifs.

Pour résoudre ce problème, une première solution est de remplacer les routes actuellement utilisées dans l'Internet par des routes offrant de meilleures performances. Toutefois, remplacer la philosophie de routage utilisée dans l'Internet semble très difficile, voire impossible en pratique. Une autre solution consiste à laisser les routes actuelles telles qu'elles sont et de réaliser du routage indirect. Considérons un chemin AB entre deux nœuds A et B. Si un chemin ACB propose un délai plus court que AB, alors C est un raccourci de routage pour AB. Dans ce cas, au lieu d'envoyer ses données directement à B, A peut les envoyer à C et C peut les relayer vers B. Procéder de la sorte s'appelle faire du *routage overlay*, car le routage est géré au sein d'un réseau overlay construit sur l'Internet. Le principal problème du routage overlay est son coût : mesurer les caractéristiques des chemins, échanger les résultats de ces mesures entre les nœuds et calculer les meilleures routes, cela requiert énormément de ressources, et une telle approche peut difficilement être envisagée à grande échelle. La contribution principale de cette thèse est la proposition d'un mécanisme de routage overlay économe en ressources. Nous proposons des critères basés sur les informations fournies par un système de coordonnées (ICS) afin d'obtenir, avec un coût minimal, un ensemble de raccourcis de routage potentiels pour n'importe quel chemin donné dans un réseau. Dans des réseaux constitués de centaines, voire de milliers de nœuds, notre meilleur critère est en mesure de réduire le nombre de raccourcis potentiels pour n'importe quel chemin donné à environ un ou deux pour cent du nombre total de nœuds.

Cette thèse présente également des résultats plus généraux concernant les raccourcis de routage et les ICS. Pour un ICS, un raccourci de routage est une violation du principe d'inégalité triangulaire (TIV) et c'est un problème majeur. En effet, les TIV n'étant pas modélisables dans un espace métrique, ils génèrent des erreurs d'estimation. Une étude des TIV présents dans l'Internet et de leur impact sur l'ICS appelé Vivaldi nous a permis d'aboutir à deux résultats. Premièrement, nous proposons des critères qui permettent d'établir s'il existe un raccourci de routage pour un chemin donné. Deuxièmement, nous proposons d'adopter une structure hiérarchique pour les ICS afin de réduire l'impact des TIV sur ceux-ci. Finalement, dans cette thèse, nous étudierons l'efficacité de deux solutions proposées dans la littérature afin d'éliminer la problématique des TIV pour les ICS. La première consiste à appliquer des transformations non linéaires aux délais avant d'essayer de les plonger dans un espace métrique. La seconde consiste à considérer le problème de l'obtention des estimations comme un problème de complétion de matrice afin d'éviter les désagréments liés au plongement dans un espace métrique.

Acknowledgements

There are several people who contributed to this work and I want to thank them all, even if I may forget some of them here.

First of all, I would like to thank my advisor, Pr. Guy LEDUC, for his invaluable support. I would like to thank him also for giving me the chance of performing this thesis. I learned a lot of things during these years and I am convinced that it will help me in the future.

I would also thank Dr. Mohammed Ali KAAFAR and Dr. Bamba GUEYE for their collaboration for a large part of the work presented in this thesis. Tips and tricks they taught me about PhD student's job were also of great help.

I thank Pr. Laurent MATHY for his collaboration during the design of the hierarchical version of Vivaldi. We shared an office during a few months at the beginning of my thesis and his support during that period has also greatly helped me.

I thank Pierre LEPROPRE for his collaboration while he was doing his master thesis. His work on the implementation was really helpful.

I thank Yongjun LIAO for her application of machine learning techniques to detect TIV bases. I also thank her for the estimations she provided me using DMF as estimation mechanism.

I thank all the people who worked in the RUN (Research Unit in Networking) team while I was doing my thesis. Even though some of them have not directly contributed to the results presented in this thesis, they all contributed to set up a very good work environment. The time shared with you during these years was great. Thank you everyone!

Last but not least, I would like to thank my family. Un très grand merci à toute ma famille pour m'avoir soutenu pendant ces quelques années de recherche. Votre attention et votre patience ont été des atouts indéniables pour la réalisation de ce travail.

This thesis has been supported by the Belgian Fund for Research in Industry and Agriculture (FRIA - Fonds pour la formation à la Recherche dans l'Industrie et dans l'Agriculture). It has also been partially supported by the EU under project FP7-Fire ECODE.

Contents

Chapter 1

Introduction

Nowadays lots of real-time applications are used in the Internet: voice over IP, online video games, *etc*. Such applications generally need some QoS (Quality of Service) guarantees and, particularly, low delays between communicating nodes in order to perform correctly. However, since the Internet was not developed with QoS guarantees in mind, the choice of the route between two nodes is not guided by QoS constraints. Thus, in many cases, there exists an alternative path providing a smaller delay than the default Internet path. Modifying Internet's routing protocols in order to choose the best routes with respect to QoS constraints should be a good theoretical solution. But, in practice, replacing the Internet's routing philosophy by a brand new one is very difficult or even impossible. Another solution, is to keep Internet's default routes as they are and to do some indirect routing. In other words, a good alternative path for a given path AB, is a path ACB so that C is a shortcut in terms of delays for the path AB:

$$RTT(A, B) > RTT(A, C) + RTT(C, B) \tag{1.1}$$

where $RTT(X, Y)$ denotes the RTT (Round Trip Time) between nodes X and Y, i.e., the time necessary to travel in the network from X to Y and back from Y to X.

In this thesis, our main goal is to find such shortcuts for any given path in a network in order to obtain smaller delays: if C is a shortcut for the path AB, we intend to use C as relay instead of sending the data directly from A to B. This is called *overlay routing* because we will manage routing between nodes in an overlay network (*i.e.* in a network laid over the Internet). Overlay routing is attractive because deploying an overlay requires only the cooperation of the nodes participating in the overlay and it only consists in deploying new software on top of existing software. Thus, overlay networks allows the deployment of new services in the Internet without cooperation of the ISPs and without expensive deployment of new equipment. This is why overlay networks have already been used for multiple purposes: to provide a multicast service [26, 13], to provide QoS to applications [78], to improve routing [1, 73], *etc*.

So, using overlay routing to improve the performance of a network is not a new idea. RON [1] (Resilient Overlay Network) was proposed in 2001 to improve the performance and the reliability of networks. The idea of RON is to build a fully connected mesh between the nodes participating in the overlay and to monitor all the paths. Using the collected information, its main goal is to allow fast path recovery thanks to indirect routing:

if a path between two nodes A and B is broken, RON tries to find a node C reachable from A and from which it is possible to reach B. In other words, its main goal is to build a resilient network on top of the Internet where path recovery can take on the order of several minutes after a failure [33]. RON's functionalities have been extended to allow also the search of indirect routes in case of performance failures: if a path between two nodes A and B has too poor performance for a given application, using a node C as relay between A and B can be the solution to provide the required QoS to the application. It has been observed in [74] that indirect routing can significantly improve the performance for many paths in the Internet. This second functionality of RON is of interest to us in this thesis.

Nowadays, even if overlay networks seem to be an easy way to improve the performance of the Internet, they are not often used by applications. The main problem is the scalability. Indeed, to know exactly which node C is the best shortcut for a given path AB, we must check if the inequality (1.1) is true for each node C. To be able to do that for any given path AB, it is necessary to measure all the paths of the overlay network and distribute measurements results among the overlay nodes. These operations become costly if a large number of nodes take part in the overlay. Indeed, if there are n nodes in the overlay, we have to measure permanently the RTT of $O(n^2)$ paths and we have to share these measurement results among the n nodes. It represents a measurement traffic which is $O(n^2)$ and a communication traffic which is $O(n^3)$. Thus, the traffic generated by the overlay is considerable and its impact on the data traffic can be significant.

A solution to avoid a too large resource consumption due to measurements is to use estimations of the RTTs instead of measuring them. The main contribution of this thesis is to show that it is feasible to use an Internet Coordinate System (ICS) [61], namely Vivaldi [15], to estimate RTTs in a scalable manner (i.e., without too much measurement overhead) and use this knowledge to find shortcuts in the network. Using the estimations provided by an ICS to detect shortcuts leads to a major problem: by definition, a shortcut situation is a Triangle Inequality Violation (TIV) that disappears after its (imperfect) embedding in a metric space [35]. Since there are no shortcuts in the feature space, equation (1.1) can never be satisfied when estimated RTTs are used instead of real RTTs. In this thesis, we will mainly investigate several ways to circumvent this problem. Intuitively, our idea consists in combining estimations with as few measurements as possible in order to be able to find routing shortcuts in a scalable manner. With our estimation-based approach [5, 8], we propose an overlay routing solution with a measurement traffic which is $O(n \times m)$ (where $m \lll n$ is the number of reference points used by each node in the ICS) and a communication traffic which is $O(n^2)$. The communication process used in this thesis to share estimations among the nodes is very basic and, as discussed in the "future work" chapter, it should be possible to reduce this traffic by at least $O(\sqrt{n})$.

The estimation-based approach of overlay routing is our main contribution. But this thesis presents also some results about ICSes and the TIVs. Before trying to detect routing shortcuts with an estimation mechanism, we will study the routing shortcuts existing in the Internet and there impact on ICSes [7]. This study leads to two contributions. Firstly, we propose a hierarchical approach for coordinate systems in order to mitigate the effect of TIVs on the estimations [29, 6]. This Two-Tier architecture is based on the clustering of nodes. Within these clusters, nodes compute coordinates to predict local distances, and keep predicting distances to nodes outside their clusters based on the original "flat" ICS. This hierarchical approach allows nodes to embed short distances (*i.e.* intra-cluster distances) with very low relative errors. Moreover, with a careful design of the architecture, it is possible to obtain this result with only a small overhead compared to the original "flat" ICS.

Another contribution developed in the light of our study about TIVs and ICSes is the proposition of criteria allowing the detection of TIV bases (*i.e.* paths for which there exists at least one shortcut) just by observing the ICS behaviour [43, 28]. For this purpose, we characterize node pairs using different metrics such that the Relative Estimation Error (REE). One of our findings is that the REE variance of TIV bases is usually smaller. This can be used to infer TIVs with some confidence without any additional measurement. Consequently, we aim at clusterizing node pairs following their REE variances using Gaussian Mixture Models (GMMs). GMMs are one of the most widely used unsupervised clustering methods where clusters are approximated by Gaussian distributions, fitted on the provided data. We apply also an AutoRegressive Moving Average (ARMA) model on the sorted REE variances to find a breaking point, because in many practical cases we cannot fit one uniform regression function to the data. Finally, instead of empirically testing different approaches to detect TIV bases, we will use machine learning techniques in order to find the more discriminative variable among many variables based on different characteristics of the paths.

Finally, this thesis will also discuss two ways proposed in the literature to obtain ICSes that are able to deal with TIVs. The first one was proposed by Wang *et al.* [82] and consists in applying non-linear transformations to delays before trying to embed them in a metric space. Since non-linear transformations eliminate TIVs, they should allow a more accurate embedding than a classical ICS like Vivaldi [9]. The second solution is called DMF and it was proposed by Liao *et al.* [42, 40]. DMF's idea consists in seeing the estimation problem as a matrix completion problem: we can build a partial delay matrix with the delays measured by the nodes between them and their reference points and use existing techniques to complete that matrix. Since this approach does not require any embedding in a metric space, it should not worry about TIVs. In this thesis, compared to a classical ICS like Vivaldi, we will evaluate the capacity of these two solutions to (i) provide better estimations and (ii) provide better shortcuts detection results.

Structure of the thesis

The thesis is organized as follows:

- In chapter 2 we will first recall some concepts about routing and QoS. Then, we will make a small survey about overlay routing related work. Finally, we will formalize the main problem we try to solve in this thesis.
- Chapter 3 begins with a state of the art about Internet coordinate systems. At the end of this survey, we will choose one ICS, namely Vivaldi, to conduct our experiments.

Then, the chapter contains a description of Vivaldi's algorithm, its parameters and some improvements that have been proposed for this algorithm.

- In chapter 4, we will investigate the problems met by classical ICSes like Vivaldi when they face TIVs. After an analysis of the TIVs existing in the Internet, we will propose some criteria allowing to suspect the presence of TIVs. As discussed in the introduction, these criteria allow to detect TIV bases with a high probability of success just by observing the ICS behaviour. This chapter also contains a little discussion about the possibility to detect routing shortcuts just by observing the ICS behaviour.
- Chapter 5 presents the main contribution of the thesis: our criteria combining estimations with as few measurements as possible in order to be able to find routing shortcuts in a scalable manner. At the beginning of the chapter, we will describe our estimation-based shortcut detection criteria and we will evaluate them using Vivaldi as estimation mechanism. These evaluation results will be compared to a random relay node selection in order to ensure that they provide significantly better detection results. Then, we propose variants of our criteria that require less measurements than the original criteria and we discuss the advantages and the drawbacks of these variants. Finally, we will try to define which characteristics the estimations must have in order to provide good detection results with our criteria. Following the conclusion of this analysis, we will try to detect shortcuts with other estimation mechanisms than Vivaldi in chapters 6, 7 and 8.
- In chapter 6, we will analyse the capacity of non-linear transformations of the delays to improve the accuracy of the estimations obtained with Vivaldi. We will also discuss the shortcut detection results obtained with our criteria when they are applied using such estimations.
- Chapter 7, briefly presents DMF (an estimation mechanism that is not impacted by TIVs) and evaluates the shortcut detection results obtained when our criteria are applied using the DMF estimations.
- In chapter 8, we present another major contribution of this thesis: the hierarchical approach for the ICSes. Firstly, we describe Two-Tier Vivaldi, the hierarchical version of Vivaldi, and we evaluate its efficiency using manually defined clusters of nodes. Secondly, we propose a self-organized distributed way to build the clusters. Finally, we evaluate the shortcut detection results obtained when our criteria are applied using the Two-Tier Vivaldi estimations.
- In chapter 9, we discuss some improvements and extensions that are potentially interesting for the work presented in this thesis.
- Finally, we will summarize our conclusions in chapter 10.

Chapter 2

Routing and Overlay routing

Abstract

Using overlay routing to find better routes in the Internet is not a new idea. The major contribution in overlay routing is RON (Resilient Overlay Networks) and it was proposed in 2001. The main problem of the RON approach is its poor scalability and, in ten years, many improvements have been proposed. In this chapter, we will first introduce the concepts of routing and overlay routing. Then, we will describe RON and different possibilities already investigated in the literature to improve RON's scalability.

2.1 Routing

The Internet is a *packet switching* network. When a user sends data to a destination through the Internet he puts that data inside *packets*, he gives a destination address to each packet and the network has to forward each packet to its destination using its destination address. Today, millions of nodes (hosts, servers, *etc*) are connected through the Internet. In such conditions, it is impossible to have a full mesh network, *i.e.* a network were a physical link is deployed between each pair of nodes. So, we have topologies like the one presented on figure 2.1. On that figure, we have hosts connected to routers that are interconnected by physical links.

Definition 1. *A* router *is a device that is connected to two or more data lines and that forwards packets between these lines. When one packet is received on one line, the router analyses the information of the packet's header in order to find its destination. Then, the router uses the information contained in its routing table in order to forward the packet on the right data line. In addition to its forwarding task, a router must also build its routing table.*

Following definition 1, the routers manage the forwarding procedure between a source and a destination through the Internet. Indeed, when a source (for example, A on figure 2.1) wants to send data to a destination (for example, B in figure 2.1), A sends its packets to its gateway (i.e. the output of the A's local network, the router that provides

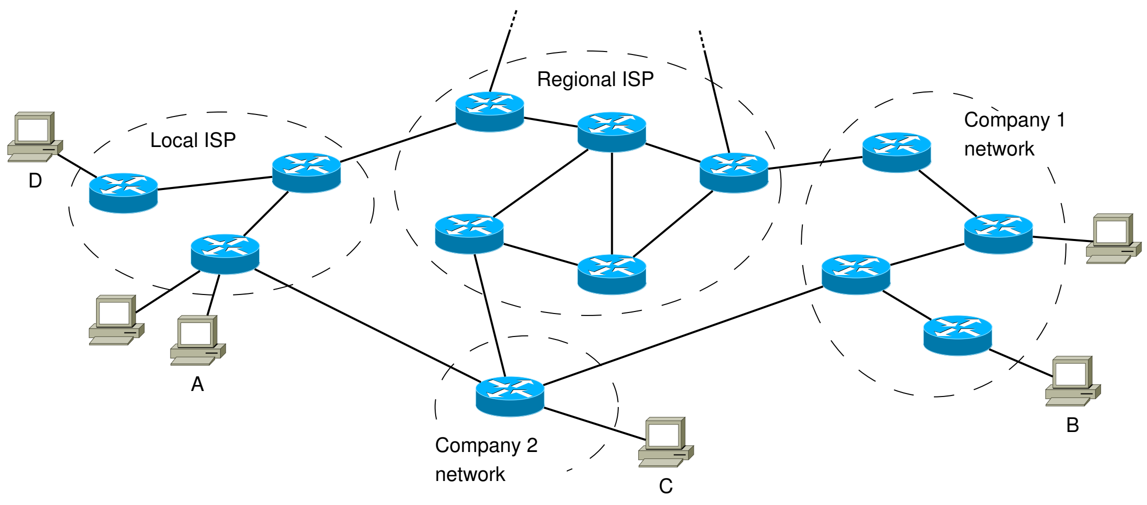


Figure 2.1: **Small example of topology.** In this figure we can see hosts connected to routers that are interconnected by physical links. Routers are grouped into Autonomous Systems (the circles on the figure). The routers that are belonging to the same AS are managed by the same entity.

an access to the Internet) and that router tries to find the next-hop router towards B using the packet's destination address and its routing table. When it finds that next-hop router, it forwards the packet to it and the next-hop router will restart the process. The process continues until the router that connect B's local network to the Internet is reached. That router will see that the packet's destination address is in a local network connected to one of its interfaces and it will forward that packet to its final destination. All that process requires routing tables to take the right decisions. These tables are built by the routers using routing protocols.

Definition 2. *A* routing protocol *is a protocol that specifies how routers must exchange information in order to be able to select a route between any pair of nodes in a network. The choice of these routes is done by* routing algorithms*.*

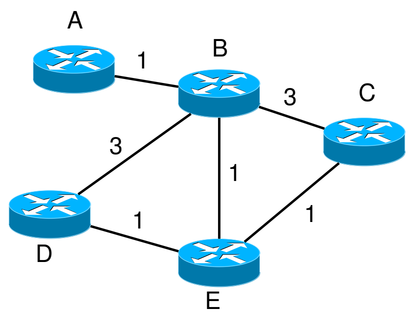
In order to keep things scalable, the Internet (and the routing) has a hierarchical structure: the routers are aggregated into autonomous systems and routers that are in a same AS run the same routing protocol.

Definition 3. *An* Autonomous System (AS) *is a connected group of one or more IP prefixes run by one or more network operators which has a single and clearly defined routing policy. [24]*

Routing protocols used inside the ASes are called Interior Gateway Protocols (IGP). Each AS runs its own IGP and, a different IGP could potentially be used in each AS. Considered individually, the IGP used by an AS allows only to find routes between the nodes belonging to the AS. In order to be able to find routes towards nodes that do not belong to the AS, it is necessary to obtain information about the topology outside the AS. In other words, it is necessary to have a routing protocol that finds routes between the autonomous systems. It is the routing protocol that is at the upper level of the hierarchical structure and it is called Exterior Gateway Protocol (EGP)[1].

2.1.1 Interior Gateway Protocols (IGPs)

Multiple IGPs coexist in the Internet. They can be classified in two categories: distance vector routing protocols (for example RIP [53]) and link state routing protocols (for example, OSPF [56] and IS-IS [63]). These protocols have basically the same objective: they try to find the shortest path between each pair of nodes belonging to a network with respect to configured link weights. The difference between them is the way they proceed to reach this objective. Our goal here is not to go into details about these protocols. We will only give a small example with OSPF using the simple topology given in figure 2.2. If the reader wants more informations about IGPs he can refer to networking introduction books like [32].



(a)

A	
B	1

B	
A	1
C	3
D	3
E	1

C	
B	3
E	1

D	
B	3
E	1

E	
B	1
C	1
D	1

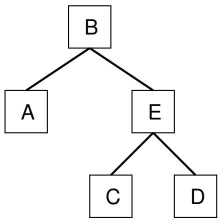
(b)

Figure 2.2: **Example of OSPF usage.** Figure 2.2(a) shows the topology with a weight configured by the network administrator for each link and figure 2.2(b) gives the link state packets broadcast by the nodes.

Consider the topology given in figure 2.2(a) which is the network of the AS of the regional ISP of figure 2.1. In order to use an IGP, it is necessary to give a *weight* to each physical link of the network. These weights are arbitrary values that are configured by the network administrator. With OSPF, each node builds its *link state packet* containing the cost of each link between the node and its neighbors. Figure 2.2(b) gives the information contained in the link state packet of each node of the topology presented in figure 2.2(a). These link state packets are broadcast through the network in order to allow each node to discover the topology of the network and the weight of each link. Thanks to these information, each node is able to compute the least cost path with respect to the weights between any pair of node in the network. In OSPF, Dijkstra's algorithm [17] is used for this purpose. That algorithm provides a *shortest path tree* that gives the least cost path between one node (the root of the tree) to all the other nodes. Each node computes its own tree by considering itself as the root of the tree and figure 2.3(a) gives the shortest

[1]We use EGP as a generic term in this document. EGP is also the name of the first exterior gateway protocol that was proposed. Today, the protocol EGP is obsolete and it is replaced by BGP.

path tree obtained for node B in our example. Using this tree, for each destination in the network, each node can decide which of its neighbors is the next-hop (see figure 2.3(b) for node B in our example). These information will be useful to build the forwarding table of the router but that is not enough: to build a forwarding table usable for any destination in the Internet, each node needs informations about the others ASes. These information are provided by an exterior gateway protocol (see section 2.1.2).



(a)

Destination:	A	C	D	E
Next-hop:	A	E	E	E

(b)

Figure 2.3: **Dijkstra's result for node B.** Figure 2.3(a) gives the shortest path tree obtained for B and figure 2.3(b) shows the information extracted from the tree that will be used to build B's forwarding table.

The main thing to keep in mind about IGPs is that they allow to optimize the routing with respect to some metrics. Indeed, each node and each link is considered individually by the protocol and the least cost path with respect to the weights is chosen between any pair of nodes in the network. So, if the weights are chosen to represent a particular metric (delay, bandwidth, *etc.*), the routing will be optimized with respect to that metric.

2.1.2 Exterior Gateway Protocols (EGPs)

The IGPs have interesting properties but they are not usable at the scale of the Internet for two reasons. The first one is the scalability. In the Internet, there are millions of possible destinations and it is impossible to store an entry for each destination in the routing tables. Moreover, with the IGPs, the nodes need to obtain a large amount of information in order to choose the least cost path to any destination. In large networks, exchanging this amount of information is too expensive[2] in terms of resource consumption. The second reason is that Internet is a network of networks and each network administrator may want to manage routing in its own network. For these two reasons, it is necessary to have a hierarchical approach of routing: each network administrator can configure its IGP in its AS to reach its own routing objective and, at the upper level of the hierarchy we have an EGP to manage the routing between the ASes. The EGP that is used today in the Internet is BGP (Border Gateway Protocol) [72]. As for IGPs, our goal is not to go into details about the

[2]With link state protocols, each node sends only information about its neighbors to the other nodes. Even if the size of the link state packets is still scalable in large networks, these link state packets are broadcast and broadcasting in large networks is expensive: in a network with N nodes and L links, the cost of broadcasting is $O(N \times L)$. With distance vector protocols, each nodes sends only its distance vector to its neighbors but that distance vector must contain the distance between the node and each destination he knows. So, with such protocols, the size of the messages exchanged between the routers is not scalable.

protocol. We will just give the bases of BGP and discuss some points that will be useful in later sections of this document.

BGP has mainly three tasks. The first one is to allow each AS to obtain reachability information from the ASes that are its neighbors: like in IGPs, routers have to exchange informations about the topology. The second one is to propagate reachability information to all AS-internal routers to allow them to build forwarding tables usable for any destination in the Internet. The last one is to determine a "good" route to each destination based on reachability information collected and some rules called *policies*.

BGP reachability information dissemation

To exchange BGP information, pairs of routers use TCP connections called *BGP sessions*. Routers connected through a BGP session are *BGP neighbors*. When the two routers connected through a BGP session belong (resp. do not belong) to the same AS, we call this session an *iBGP session* (resp. *eBGP session*). Inside an AS, an iBGP session is established between each pair[3] of routers that belong to the AS. The BGP sessions established for the example given in figure 2.1 are given in figure 2.4.

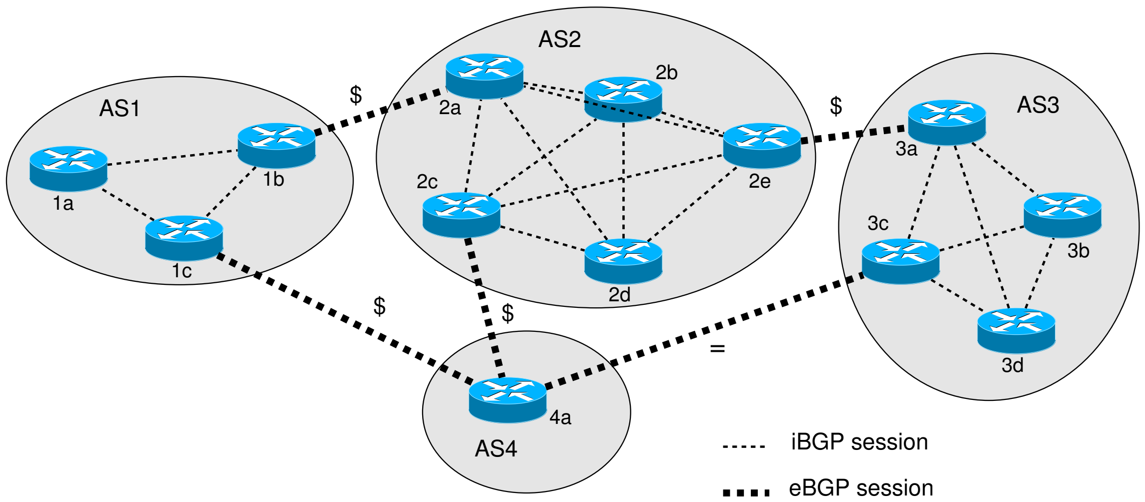


Figure 2.4: **BGP sessions established between the routers given in figure 2.1.** An eBGP session can be established between a customer AS and a provider AS (symbol "$") or between peers (symbol "=").

Once BGP sessions are established, BGP neighbors exchange information about the AS topology. BGP is a *path vector protocol*: each router sends its path vector to its neighbors. For each AS for which a router agrees to relay traffic, the path vector of that router contains the AS path it uses to reach that AS and some information to allow other routers to choose between that path and other paths potentialy received from their other neighbors. For example, in figure 2.4, AS_1 advertises reachability information to AS_2

[3]This is for the basic version of BGP. In practice, requiring a full-mesh is a real problem in term of scalability and more advanced solutions have been proposed (using route reflectors and/or defining confederations).

through the eBGP session established between routers 1_b and 2_a: AS_1 advertises to AS_2 that the addresses owned by AS_1 are reachable through him with the AS path "AS_1" and, since AS_4 is an AS_1's customer, AS_1 also advertises to AS_2 that the addresses owned by AS_4 are reachable through him with the AS path "AS_1, AS_4". When the router 2_a receives the path vector, it disseminates the reachability information in its AS using the iBGP sessions and AS_2's nodes complete their forwarding tables. Moreover, routers 2_c and 2_e advertise new reachability information to their neighbors belonging to other ASes.

Routers advertise reachability information to their neighbors but they do not advertise all the routes to all neighbors. The general rule is that an AS wants to route only to/from its customers. Have a look on the AS_4 in figure 2.4. AS_4 is a company network that has two providers: AS_1 and AS_2. The eBGP sessions between these ASes are *customer-provider* sessions (denoted by the symbol "$"): sending data on the corresponding links has a cost for AS_4 and, logically, AS_4 do not want to relay traffic coming from one of its providers to another of its providers. Consequently, AS_4 advertises AS_1 (resp. AS_2) that it has a path to AS_4 but it does not advertise the path to AS_2 (resp. AS_1). Moreover, in addition to the customer-provider sessions, ASes may also establish *peering* sessions (denoted by the symbol "="). A peering relationship is an agreement between two ASes in order to exchange traffic between them (and their customers) without using their respective providers. For example, on figure 2.4, a peering relationship is established between AS_4 and AS_3. Through the corresponding link, AS_4 (resp. AS_3) can only send traffic destined for AS_3 (resp. AS_4). Consequently, with the eBGP session between router 4_a and router 3_c, AS_4 does not advertise the paths to AS_1 and AS_2.

BGP routing policies

When a router learns multiple paths from its neighbors for a same destination, it has to choose one of these paths. For example, in figure 2.4, the routers belonging to AS_2 receive two paths for the addresses of AS_4: AS_4 advertises the AS path "AS_4" for that destination and AS_1 advertises the path "AS_1, AS_4" for that destination. In the absence of policy, a router would choose the shortest path with respect to the number of AS-hops. In our example, the routers of AS_2 would choose to route the traffic destined for AS_4 to the router 2_c and that router would forward that traffic directly to AS_4. However, in order to give more control to the network administrators over the selected paths, some attributes are added to the advertisements sent by routers and multiple rules to choose a path are defined using these attributes.

The BGP decision process consists of an ordered list of characteristics across which paths are compared. When a router learns two paths for a same destination, the router goes down the list and compares the two paths for each characteristic specified in the list. If the paths have different values for one characteristic, the router choose the path that has the most desirable value. Otherwise, it goes down in the list and compares the paths with respect to the next characteristic given in the list.

Suppose that a router learns multiple paths to a same destination. The first comparison point between the paths is the *LocalPref* attribute. When a router learns a path from a neighboring AS, it sets the *LocalPref* attribute (an integer) of that path and it advertises that path to the other routers of its AS with that attribute. The value of that attribute is deleted before advertising the neighboring ASes: the *LocalPref* attribute is a local value fixed using local policies. The router chooses the path which has the higher value for its *LocalPref* attribute. If some paths have the same value for the *LocalPref* attribute, then the router chooses the path that has the lowest AS path length. If some paths also have the same length, some rules based on the attributes of the paths are defined to choose one of them. These rules are well defined in BGP but we will not enter into details about them. We will only discuss a general principle used in BGP: the hot potato routing. The *hot potato routing* is the principle of sending the traffic destined for another AS as fast as possible outside the AS. Consequently, with hot-potato routing, a router will choose the path which has the smallest IGP cost to exit its AS. With that behavior, an AS minimizes the resource consumption for outgoing traffic.

BGP and performance

In section 2.1.1, we saw that IGPs may be configured to optimize the routing with respect to some metrics. For example, if the weights of the links are delays, the route selected between any pair of node will be the route with the smallest delays. With BGP, it is not the case because policies dominate performance. Indeed, the selection of the routes in BGP is based on commercial rules rather than on performance criteria: some paths are not advertised by the ASes to their neighbors, an AS administrator may define policies to force BGP to choose another path than the shortest one in terms of AS-hops (with the *LocalPref* attribute) and, even if the path with the smallest number of AS-hops is used between a pair of nodes, there are no guarantees that this path is the best with respect to a given metric (delay, bandwidth, *etc.*). Consequently, at the level of the inter-domain routing, there may be some paths that are better with respect to a given metric than the one chosen by BGP. We will investigate that in section 4.3.1.

2.1.3 Quality of Service (QoS)

When the routing protocols presented in sections 2.1.1 and 2.1.2 were deployed, the main usage of the Internet was different from what we need today. At this time, the Internet was mainly used to get simple html pages or to send e-mails. In such context, the most desirable property of a network is reachability: any destination must be reachable from anywhere. The routing protocols ensure reachability with a re-convergence time that can be more or less long in case of modification of the topology (for example a router or a link failure). So, these routing protocols were sufficient. But today, the usage of Internet has changed. We use it for multimedia applications: for IP telephony, to play online video games, to watch videos (streaming), to organize video conferences, *etc.* For such applications, ensuring reachability is not sufficient. They need more guarantees on the characteristics of the path used between communicating nodes. Indeed, to perform correctly, some of them require a small delay, others require a large bandwidth, others require a small delay and a large bandwidth and others can also require guarantees on more specific characteristics (delay jitter, *etc.*). In other words, lots of applications deployed today in the Internet require some level of quality of service.

Definition 4. *The* Quality of Service (QoS) *is the ability to provide different priorities to different applications, users, or data flows, or to guarantee a certain level of performance to a data flow.*

Basically, the Internet operates on a best-effort basis. During the forwarding procedure, packets are processed in the arrival order by the router and the service is the same for each packet. For example, a packet belonging to a flow generated by a VoIP (voice over IP) application is processed the same way than a packet generated by the download of a file. Ideally, since delays are critical for VoIP, the packets generated by that application would have to be processed in priority. To solve such problems, QoS mechanisms have been proposed for the Internet. We will give a short description of three common approaches used to provide QoS in the Internet. Since these notions are a little bit out of scope, we will not enter into details about them. If the reader wants more information on the subject, he may refer to the literature [32, 31].

Intserv (Integrated Services)

A first solution proposed to provide QoS guarantees in the Internet is Intserv [3]. With that approach, applications use RSVP (Resource ReSerVation Protocol) [4] to request and reserve resources on the path between the source and the destination of the flow. Before sending the data, there is a call admission phase. During that phase, the application declares its needs to the network components and the network can reject the call if it cannot meet these needs. If the call is accepted, since the resources are reserved, the application has the guarantee that its packets will use a path that has characterics corresponding to the its requirements. However, that approach has a problem of scalability because the Internet's core routers would have to keep state information for each reservation. Consequently, Intserv is an interesting approach because it is a fine-grained QoS mechanism where each flow does its own reservation depending of its own requirements but it is only usable at a small scale.

Diffserv (Differentiated Services)

A second approach to provide QoS in the Internet is Diffserv [62]. The basic idea of Diffserv consists in defining multiple classes of traffic and processing the different classes differently in the network. A simple example is to define two classes "prior" and "best-effort" and packets marked as "prior" have to be processed in priority when they are received by a router. This is only the basic idea: in reality, more than two classes are generaly defined and more or less complex mechanisms (policing, shaping, etc) are necessary to have something usable. Following that small description, Diffserv is a coarse-grained QoS mechanism (it provides a per-class resource reservation mechanism and not a per-flow reservation mechanism) but it is more scalable than Intserv.

Over-provisioning

Instead of deploying complex QoS mechanisms, an alternative to provide a quality environment to applications consists in deploying a physical network that is able to support any traffic load. This is a simple solution that provides a reasonable level of performance for lots of applications but it does not really solve the problem of QoS. Indeed, this principle assumes that the network has "enough" capacity to support any traffic load. In such conditions, what is "enough" ? For example, if you provide more bandwidth to a TCP flow, its bandwidth consumption will increase until it will begin to lose packets. So, if the network provides more capacity, the TCP flows will potentially adapt their bandwidth consumption in order to use all the available bandwidth. So, whatever large capacity you give to your network, it is always possible to find situations where you will experience congestion and where the "natural" QoS provided by this approach disappears. Moreover, the level of performance reached with that approach is sufficient for many applications but not for all applications. Indeed, even if the network has a large capacity, the applications' packets can still experience potentially large delays and bandwidth variations. For example, a packet generated by a VoIP application may still reach a router just after a large burst generated by a TCP best-effort flow. If the router does not manage any notion of priority, the VoIP packet will have to wait the processing of all the packets of the TCP burst before being processed. Consequently, over-provisioning is a suitable approach for applications that can compensate delay and bandwidth variations like video streaming (by using large buffers at the receiver) but is not a suitable approach for other real-time applications like VoIP.

2.1.4 Routing on performance basis

The QoS mechanisms presented in section 2.1.3 have one major drawback: the routing mechanism and the QoS mechanism are independent. So, if the routing mechanism chooses a route with bad characteristics between two nodes, the QoS mechanism can try to reserve whatever it wants as resources on that path, it will never be able to meet some application QoS requirements. For example, if the route chosen between two nodes A and B is such that the minimal delay between A and B is $500\,ms$, the QoS mechanism cannot guarantee a path with a delay of less than $500\,ms$ between A and B. But $500\,ms$ is a lot too much for applications like VoIP. In such cases, the only solution consists in finding another route between A and B.

To be able to find good routes from the QoS point of view, the best solution would probably be to replace the existing routing mechanism by another one that would take performance into consideration. However, such approach induces several problems:

1. The characteristics of a "good route" are different for each application. For example, a VoIP application requires only constraints on delays when a videoconference application requires also constraints on bandwidth. In such conditions, choosing routes suiting all applications is difficult.
2. Having a routing mecanism that computes the best routes using a variable metric like the delay or the available bandwidth is difficult to manage. When a route

between two nodes is chosen by a routing mechanism, that route will support an additional traffic load and its characteritics will change. The new characterics of that route may become inadequate for some applications for which another route may be suitable.

3. Using that approach for the Internet means modifying the routing philosophy of the Internet. But in a network, everybody must use the same rules to choose the routes. If it is not the case, there may be loops, some destinations may be unreachable, etc. In other words, replacing existing routing protocols by new ones is a tricky operation that can lead to several problems.

Since modifying the existing routing protocols is difficult to manage, a solution consists in keeping the current routes for the best-effort traffic and computing new routes for the traffic that needs it. Overlay routing (routing in an overlay network) is a common approach to reach such a goal without adding complexity to the current routing mechanisms.

2.2 Overlay routing

Overlay routing refers to the usage of an overlay network to provide a particular routing behavior.

2.2.1 Overlay Networks

Definition 5. *An* overlay network *is a network built on top of another network. The edges between the nodes in an overlay network are paths (that can be composed of many physical links) in the underlay network.*

Figure 2.5 gives an example of an overlay network built on top of the network given in figure 2.1. For this overlay network we used the four end-hosts named A, B, C and D in figure 2.1. Each of these end-hosts is a node in the overlay network. On the left part of figure 2.5, we see that four links are defined between these nodes. These links are virtual links. Indeed, each of these links corresponds to one path in the underlay network. The paths in the underlay network corresponding to each virtual link of the overlay network are given in the right part of figure 2.5. For this example, we used only end-hosts as nodes but other network components (*i.e.*, routers, servers, *etc.*) can also be used as nodes in an overlay network.

Overlay networks are used to provide services that are not provided by the underlay network. The usage of overlay networks to deploy new services has mainly three advantages compared to a direct deployment in the underlay network:

Easy deployment: Deploying an overlay network means deploying new software on top of existing software. It does not require an expensive deployment of new equipment or a potentialy dangerous modification of existing protocols. Since the risks are limited (from an economical and a practical point of view), there is a natural incentive to deploy such solutions.

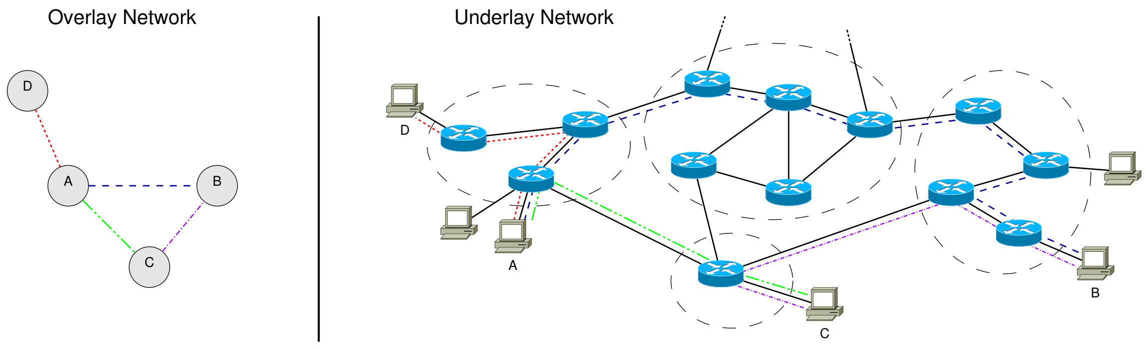


Figure 2.5: **Example of overlay network built on top of the network given in figure 2.1.** The left part of the figure gives the topology of the overlay network (four nodes connected by virtual links) and the right part of the figure shows the path in the underlay network corresponding to each virtual link.

Progressive deployment: The deployment of a new service requires the development of a communication protocol between the nodes of the network. For example, to deploy a new routing protocol, all the routers have to understand the new messages received from their neighbors. Consequently, all the nodes have to be updated simultaneously and that can prevent the deployment of the new service. If that service is provided using an overlay network, that problem disappears. Indeed, nodes in an overlay communicate through virtual links and do not require the compatibility of all the routers standing on the physical paths used by the virtual links. In such conditions, it is possible to start working with only a few nodes and adding progressively new nodes later. In other words, it is not necessary to modify the entire network at the beginning and the bootstrap phase of new protocols and services is a lot easier.

Partial deployment: The source of this third advantage is the same as for the previous one: deploying a new service using an overlay network approach does not require the cooperation of all the routers standing on the physical path used by the virtual links defined in the overlay. Consequently, it is possible to deploy new services without the cooperation of all the ISPs (Internet Service Providers) acting in the Internet. That was the major drawback of Diffserv and Intserv: in order to obtain QoS guarantees between a source A and a destination B, all the routers on the path between A and B have to be Intserv/Diffserv compliant. Since some ISPs are not convinced by the necessity of such protocols, they are not widely deployed and there are lots of situations were they are not usable. With an overlay containing only nodes that are end-hosts (like in figure 2.5), it is possible to provide a new service to the Internet users without any cooperation from the ISPs.

Overlay networks have been already used for multiple purposes. Among others, they have been used in the Internet to provide a multicast service [26, 13], to provide QoS to applications [78] and to improve routing [1, 73]. That last application of overlay networks is called *overlay routing* and is the one that interests us.

2.2.2 Detour

Detour [73] was the first major contribution in overlay routing. In [74], Savage *et al.* used the results presented by Paxson [65, 66] and their own measurements results to show that, for a large number of paths in the Internet, it is possible to find an alternative path providing better performance. Following that observation, they proposed Detour. The Detour's architecture is composed of intelligent routers deployed at key access and interchange points in the Internet and the traffic is sent through virtual links established between these nodes. That architecture allows to improve the routing performance (the routers exchange measurement results in order to choose the best route in the overlay), to propose multipath routing possibilities, to improve the efficiency of TCP (by providing better information about the network's state to the transport protocol), *etc.*

Detour is a service deployed inside the Internet. Indeed, the nodes belonging to the overlay are routers and not end-hosts. If an application running on an end-host wants to use the Detour service, it has to send its outgoing traffic to the nearest Detour router. Then, the packets are forwarded along the virtual links of the overlay and exit the Detour network at a point close to the destination. That design has mainly two drawbacks. Firstly, the solution is application-independent and does not allow an application to define quality metrics and routing decision rules depending of its specific requirements. Secondly, the deployment of new routers which are Detour compliant (or the adaptation of existing routers) in the heart of the Internet requires the cooperation of ISPs. However, ISPs will probably not be interested in the deployment of such service. Indeed, even if it has been observed that such service can provide better performance from an end-user point of view, we will see in section 4.3.1 that, in many cases, the alternative paths proposed by such services violate the routing rules applied by the ISPs. These drawbacks can explain the low success of Detour compared to solution proposed two years later: RON.

2.2.3 Resilient Overlay Network (RON)

The main goal of RON [1] is to enable to recover from failures in the Internet within several seconds while BGP may take in the order of several minutes to converge to a new valid route after a link failure [33]. For applications, there are two types of failures: there are *outages* where some destinations are unreachable and *performance failures* where the path chosen by the Internet to reach some destinations has (perhaps temporarily) too bad characteristics to meet the application's requirements. RON intends to detect these failures and to find alternative routes that circumvent the detected problems.

To reach its objectives, RON builds a full-mesh overlay network (*i.e.* a network where a virtual link is defined between each pair of nodes) composed of application-layer overlay nodes deployed on end-hosts. In order to detect failures as fast as possible, RON nodes monitor permanently the reachability and the quality of the Internet paths corresponding to the virtual links established between them. The results of the monitoring done by each node are distributed among all the nodes to allow them to use this information to decide to route packets directly over the Internet or indirectly through other nodes. Since nodes are end-hosts, each node can take its own decision based on its specific requirements.

In [1], Andersen *et al.* observed that, over a 64-hour period of monitoring in a RON network composed of twelve nodes in the Internet, there were 32 significant outages and RON's routing mechanism was able to detect and route around all of them in less than twenty seconds on average. Furthermore, they observed that RON was able to improve the loss rate, latency, or throughput perceived by data transfers. Finally, they observed that forwarding packets via at most one intermediate RON node is sufficient to overcome faults and improve performance in most cases. They concluded that RON's approach for fault detection and recovery works well at discovering alternate paths in the Internet.

Comparison with Detour

RON shares with Detour the idea of routing through intermediate nodes but differs from Detour mainly on two points. Firstly, RON is designed as an application-controlled routing overlay while Detour is application-independent. Consequently, RON can integrate specific metrics and policies required by applications. Secondly, RON can be more easily deployed than Detour (it requires a deployment on end-hosts while Detour requires a deployment on routers). Finally, RON gives experimental results obtained from a real-world deployment.

RON's scalability

Even though RON's approach has interesting characteristics and a small deployment in the Internet has given good results, RON has a big problem of scalability. Indeed, RON has a huge measurement cost and a huge dissemination cost.

Definition 6. *The* measurement cost *denotes the traffic load generated by the measurements required by the overlay routing mechanism. The* communication cost *denotes the traffic load generated by the exchanges of information between the nodes participating in the overlay routing mechanism.*

Measurement cost: With RON's approach, each node in the overlay has to perform permanent measurements with all the other nodes in order to enable a quick detection of failures. Since RON's overlay network is a full-mesh network, there are $n \times (n-1)/2$ bidirectional virtual links to monitor in a network containing n nodes. Consequently, RON has a measurement cost which is $O(n^2)$.

Communication cost: With RON, each node has to send its measurement results to each other nodes. So, in a network containing n nodes, the number of messages is $O(n^2)$. Since each message contains the measurements computed between the sender of the message and all the other nodes of the network, the size of each message is $O(n)$. Consequently, the traffic generated by the dissemination of the measurement results among the nodes is $O(n^3)$.

Notice that these estimations of the costs are done for the monitoring of one parameter. Indeed, if the overlay requires the monitoring of p parameters (bandwidth, latency, *etc*), the measurement cost becomes $O(p \times n^2)$ and the communication cost becomes $O(p \times n^3)$.

2.2.4 Improving RON's scalability

With a measurement cost $O(n^2)$ and a communication cost $O(n^3)$ there is room for improvements and, over the years, many improvements have already been proposed for RON's approach. They can be grouped into three categories: some improvements try to reduce the number of virtual links to monitor in the overlay, others try to reduce the communication overhead and, finally, others try to reduce the measurement overhead.

Elimination of redundant overlay links

A first approach to improve the scalability of RON's approach consists in reducing the number of overlay links to monitor. With that approach, the measurement cost is reduced and the communication cost is reduced (since there is less information to exchange among the nodes).

Nakao *et al.* [57, 58] proposed to eliminate redundant overlay links by using topological information in order to build a routing mesh that is representative of the underlying physical network. So, by contrast to many overlay mechanisms they do not consider the Internet as a black-box and they try to eliminate virtual links that use the same physical links in the underlay. To provide a scalable overlay routing mechanism, they use only passive measurements and topological information that rarely change (such as AS-level topology and geographical information) to build their overlay network. Their experiments showed that their approach is able to reduce the measurement cost and the communication cost by a factor of two with only a small negative impact on route selection. Fei *et al.* [20] used a similar way to select, for each pair of nodes of a network, a small number of AS-disjoint paths as potential alternative paths.

Another way to reduce the number of links to monitor consists in replacing RON's unstructured overlay by a structured overlay. As the routing in the Internet uses a structured architecture to be scalable, Qazi and Moors [70] proposed to do the same in the overlay, namely to define logical zones built around landmarks and to attach each node to one logical zone. In that structure, each node monitors only a few nodes selected in each logical zone.

Reducing the communication overhead

Another solution to improve RON's scalability consists in reducing the communication overhead generated by the exchanges of the measurements results between all the nodes. In RON this overhead is $O(n^3)$ where n is the number of nodes in the overlay. Sontag *et al.* [77] proposed a protocol allowing each node to find an optimal one-hop path to any other node with a communication cost $O(n^2 \times \sqrt{n})$ and to find the optimal path to any other node with a communication cost $O(n^2 \times \sqrt{n} \times log(n))$. Let's consider only the search of one-hop alternative paths (this is a little bit more complicated with multi-hop alternative paths). Their approach is based on the fact that, if a node C receives the results of the measurements done by A and the results of the measurements done by B, C is able to find the optimal one-hop path between A and B. Consequently, it is not necessary to disseminate the measurement results in the whole network and each node can only send its measurements results to a few other nodes.

Reducing the measurement overhead

A last common way to improve the scalability of RON's approach is to reduce the measurement overhead. Since the objective is to change traffic routes, we suppose that measurements must be done quite frequently to have accurate information about the state of the network. Indeed, even if some studies [84] have shown that metrics like loss, latencies and throughput remain generally constant over time intervals of a few minutes, the goal of a system like RON is to detect and recover from failures as fast as possible. Consequently, monitoring only one time every 5 minutes (for example) is not sufficient for RON and the network has to be monitored permanently[4].

To circumvent this problem Gummadi *et al.* [22] proposed to route through random relay nodes instead of doing measurements and they observed that it is sufficient to ensure reliability. However, Sontag *et al.* [77] observed that it is not sufficient to find good alternative paths considering particular metrics like latency.

Recently, Lumezanu *et al.* [48] proposed to use an ICS (Internet Coordinate System) to find alternative paths in a network. An ICS is a mechanism that allows to estimate metrics like latency between any pair of nodes of a network with a small measurement cost. However, we will see in chapter 4 that such mechanism is unable to provide accurate estimations when an alternative path has better characteristics than the direct path between two nodes. In other words, when there exist routing shortcuts, there are estimation errors. Lumezanu *et al.* proposed to infer the existence of routing shortcuts from the observation of these estimation errors.

2.3 Problem statement

Consider that an application wants to send data from a node A to a node B with the constraint that $RTT(A,B)$ must be smaller or equal to some threshold x. If $RTT(A,B) > x$, the application cannot perform correctly. In such case, we intend to provide a service that can be used by the application: it can ask to find an alternative path to go from A to B with the smallest possible delay. When such request is received, we will try to find the best C node such that $RTT(A,C) + RTT(C,B) < RTT(A,B)$. Such service is already proposed by existing overlay routing systems like RON (RON proposes even more than that service). However, RON is a measurement-based service and is not scalable. Our objective is to provide that service in a scalable manner by using an estimation-based approach instead of using a measurement-based approach: instead of performing measurements between A and each potential C and between B and each potential C, we intend to use an ICS to obtain estimations of the required RTTs. If we find at least one node C which is

[4]As indicated by some studies (*e.g.* [77]), measurements and route computations could be done, for example, every 5 minutes and failures could be discovered through passive measurements (triggering an active monitoring phase and a route recomputation phase when a failure is discovered). Indeed, since a failure is problematic only if it impacts a flow, detecting the failures by analysing the flows is sufficient.

a routing shortcut for the path AB, the best shortcut is returned to the application. Then, the application can use that node as relay to route indirectly its data from A to B with a smaller delay than the delay experienced using the Internet default path.

In this thesis, our main objective is to prove the feasibility of using a scalable estimation-based approach to provide the same service as RON's non-scalable measurement-based approach.

Before going deeper into the subject, we want to clarify some important points about our research:

We focus on RTTs as performance indicator: RON considers more metrics but RTT seems a good choice for a proof of feasibility, because it is simple to monitor and sufficient for many applications (see section 3.1).

We are only interested in RON performance improvement functionality: We do not intend to provide a resilient overlay network (RON's main objective). Providing such service with an estimation-based approach is simply impossible. Basically, the principle of an ICS is to measure only the RTTs of a few paths in a network and to infer estimations of the other RTTs from these measurement results. So, the first problem with an estimation-based approach is that the system will only be able to detect outages for the measured paths (if everything is "normal" for the measured paths, it is impossible to infer from the measurements that there is an outage on an estimated path). Even if an outage is detected for a measured path, the second problem is that it is impossible to know if estimated paths are impacted or not by the outage. So, even if the failure is detected, using only the estimations, it is impossible to find a C node allowing to circumvent the outage. In practice, the ICS will simply wait for the establishment of new routes by Internet's routing protocols and then reconverge to estimations reflecting the new RTTs. Note that the problem does not appears for RON's performance improvement functionality. Indeed, in case of a simple performance failure, all the paths can still be monitored and the ICS can immediately reconverge to new estimations reflecting the performance changes.

We do not intend to improve all the routes in the Internet: Previous studies [30] have shown that uncoordinated effort between ISP's traffic management and overlay's traffic management may cause performance degradation for both overlay and underlay traffic. That's essentially why we do no intend to optimize the routes for the whole traffic: an application can ask a routing shortcut when it is necessary[5] (*i.e.* when the default route has too poor performance) and only the packets generated by that application will be routed through the alternative path. In our mind, the best effort traffic still uses the default Internet routes. This ensures that the major part of the traffic remains unchanged. Thus, we avoid the problems presented in [30].

We focus on one-hop routing shortcuts: Previous studies (*e.g.* [51]) have shown that limiting alternative routes to one intermediate hop is sufficient to solve many performance failures. The improvement provided by multi-hops alternative routes is

[5]We can imagine a pay per use service in order to avoid systematic requests from applications.

quite small compared to the additional costs induced by the consideration of such paths. However, the possibility to extend our work to multi-hops routing shortcuts will be briefly discussed in section 9.3.

Each overlay node is a potential shortcut: In a recent study, Lumezanu *et al.* [51, 48] have investigated the problem of mutual advantage in overlay routing: they consider that overlay edges should exist only between hosts that benefit from each other's resources or position in the network. Obviously, that reduces the number of potential shortcuts for each path. However, their study has shown that, even with the constraint of mutual advantage, it is still possible to find routing shortcuts for a large part of the paths in the Internet. For our work, we do not consider such constraint and any overlay node can potentially be a shortcut for any overlay link.

2.4 Conclusion

In this chapter, we introduced the concept of overlay routing. Using overlay routing to provide a better routing service than the one provided in the Internet seems promising mainly because it can provide the required service (for example, refer to the experiments presented in [1]) and it is quite easy to deploy (because it requires only the cooperation of the nodes participating in the overlay). However, the overlay network approach is not often used by applications because of its poor scalability. RON [1] was the first main contribution in overlay routing and it was proposed in 2001. Since 2001, many improvements have been proposed in order to increase its scalability. However, none of them has really convinced. Recently, [48] proposed to use an ICS to estimate the characteristics of the paths instead of measuring them, but their usage of the ICS was quite basic. We think that the usage of an ICS to replace the measurements is an interesting approach: it can reduce drastically RON's measurement and communication costs. Obviously, these cost reductions will have consequences. The main drawback is that we will lose precision because we will use estimations of the characteristics of the paths instead of real measurements. This can lead to detection errors (thinking that an alternative path has good characteristics when this is not the case) and to suboptimal solutions (finding one good alternative path which is not the best one). Even if the obtained results are perfectible, we think that such approach can give good results with the major advantage of being scalable. In the following chapter, we will introduce the concept of ICS and, in the next chapters, we will investigate ways to use the estimations produced by an ICS to detect alternative routes.

Chapter 3

Internet coordinate systems

Abstract

In the previous chapter, we have seen that overlay routing is a promising way to improve the quality of the routes in the Internet, but we have also seen that the scalability is a major drawback with that approach: a huge amount of traffic is generated by the monitoring of the network and by the dissemination of the measurements results among the nodes. In this chapter, we will introduce the notion of Internet Coordinate System. Such systems allow for the estimation of metrics between any pair of nodes in a network with only a small measurement cost. Consequently, Internet Coordinate Systems seem to be a simple way to solve the problem of scalability of overlay routing, but we will see in the next chapter that this is not so simple.

3.1 Round Trip Time as performance indicator

Our objective is to find a lightweight mechanism that provides indications about the performance of the paths in a network. Ideally the performance of a network path should be represented by the QoS perceived by the users of the path. However, such metric depends of the user requirements, is difficult to quantify and requires the intervention of the users to be quantified. We need something more convenient as metric. In the following sections of this thesis we will focus on one metric: the Round-Trip-Time (RTT).

Definition 7. *The* Round Trip Time (RTT)*, is the time required for a packet to travel from a specific source to a specific destination and back again. In the Internet, this can be measured by computing the time elapsed between sending an ICMP echo request packet to the destination and receiving the corresponding ICMP echo reply packet.*

This choice has been done mainly for three reasons:

- RTT is a simple metric useful for many applications. For example, VoIP (Voice over IP, *i.e.* IP telephony) can be highly impacted by a large RTT between communicating nodes. Another example is online realtime multiplayers video games where a small RTT between players is required for a good experience. For other applications the RTT is an important metric, but not the only one. For example, video-conferences require small delays between communicating nodes but also a large bandwidth for the video signal. Anyway, in lots of cases, the RTT is an important indication of network performance.

- Compared to other significant metrics of network performance, the RTT is easy the monitor. For example, compared to available bandwidth measurements, RTT measurements consume a lot less network resources and have less impact on the ordinary traffic. Compared to one-way delay measurements, RTT measurements do not require synchronizations between nodes (which is difficult to obtain).
- Several Internet Coordinate Systems have already been developed to provide RTT estimations.

3.2 Internet Coordinate Systems

An *Internet Coordinate System (ICS)* is a mechanism that allows network nodes to estimate metrics (like latency or available bandwidth) between them without performing direct measurements. To achieve that goal, an ICS models the network as a geometric space and computes a position for each node in that space. The position of a node in the space is defined by its *coordinate*. These coordinates are such that the distance between the coordinates of two nodes gives an estimation of the metric considered by the ICS between these two nodes. For example, in figure 3.1 the ICS models the network as a two-dimensional Euclidean space to estimate latencies. Nodes A and B have respectively the coordinates (x_a, y_a) and (x_b, y_b) in that space. An estimation $\hat{L}(A, B)$ of the latency $L(A, B)$ existing between the nodes A and B can be computed as the Euclidean distance between the coordinates of these nodes:

$$\hat{L}(A, B) = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}$$

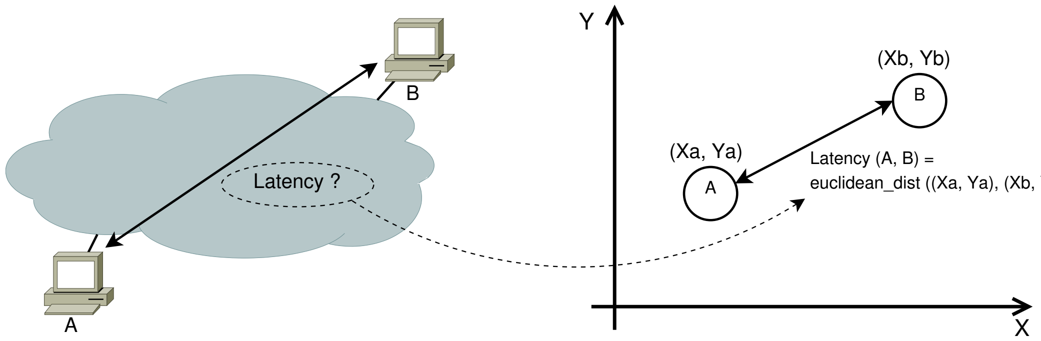


Figure 3.1: **Example of ICS usage.** This ICS models the network as a 2-dimensional Euclidean space to allow latency estimations. The latency between two nodes is estimated by the Euclidean distance between the coordinates of these nodes.

In figure 3.1 we took as example an ICS that estimates latencies. This choice was not done randomly. Indeed, most of the ICS (*e.g.* [61, 15, 14]), have been developed to provide latency estimations. However, recent works (*e.g.* [71, 41]) have shown that it is also possible to provide estimations for metrics like the available bandwidth.

It is important to notice that the estimation of a metric between two nodes with an ICS requires only the knowledge of the coordinates of these nodes and can be computed even if it has never been physically measured. In fact, the global objective of an ICS is to minimize the estimation errors with the constraint that the network resource consumption must be as small as possible.

Definition 8. *Let $RTT(A, B)$ be the measured RTT between the nodes A and B and let $EST(A, B)$ be the estimated RTT between the nodes A and B using the coordinates of these nodes. The* Absolute Estimation Error $\mathrm{AEE}(A, B)$ *and the* Relative Estimation Error $\mathrm{REE}(A, B)$ *for the Internet path between the nodes A and B are*

$$\mathrm{AEE}(A, B) = EST(A, B) - RTT(A, B) \qquad\qquad \mathrm{REE}(A, B) = \frac{\mathrm{AEE}(A, B)}{RTT(A, B)}$$

Following these definitions, $\mathrm{AEE}(A, B)$ and $\mathrm{REE}(A, B)$ are negative (resp. positive) values when $EST(A, B)$ underestimates (resp. overestimates) $RTT(A, B)$.

Obviously, measuring the path between each pair of nodes and applying an optimisation algorithm to compute the best coordinate for each node will provide an optimal result, but is not an acceptable solution. Indeed, if a measurement result is available for each path, the estimations become useless. Consequently, an ICS must only measure a few paths in the network and infer the position of the nodes in the space from the measurement results and the positions of the other nodes in the space. There exist multiple ICS and each ICS has its own method to compute the coordinates of the nodes. A good survey on this subject can be found in [18]. Based on their general behavior, the coordinate systems can be divided into two categories: the *landmark-based* coordinate systems (see section 3.2.1) and the *decentralized* coordinate systems (see section 3.2.2).

3.2.1 Landmark-based coordinate systems

In a landmark-based coordinate system, there are generally two categories of nodes: the *landmarks* and the *ordinary nodes*. In a classical scheme, each landmark performs measurements with all the other landmarks in order to compute a coordinate that gives it a good position in the space considering the position of the other landmarks. Then, when an ordinary node wants to compute its coordinate, it performs measurements with the landmarks in order to find a good position in the space considering the position of the landmarks.

State of the art

Global Network Positioning (GNP) [61] was the first mechanism that proposed to model the Internet as a n-dimensional geometric space. GNP implements the basic behavior for landmark-based coordinate systems described above. In GNP, only a small number of nodes are designated as landmarks. Each landmark performs measurements with all the other landmarks and a position in the geometric space that minimises estimation errors on the measured paths is computed for each landmark (using an optimization algorithm called the Simplex DownHill method [59]). When an ordinary node wants to compute its coordinate, it performs measurements with respect to each landmark and uses the Simplex DownHill method to find a coordinate that minimises the estimation errors for the paths between it and the landmarks.

Lighthouses [68] is a GNP variation. With GNP, each ordinary node must do measurement with each landmark. So, when the number of ordinary nodes grows, the incoming measurement traffic at the landmarks grows proportionally. That becomes a real problem if GNP is deployed at a large scale. To circumvent this problem, Lighthouses proposes to use multiple landmarks sets. When an ordinary node wants to compute its coordinate, it performs measurements only with the landmarks belonging to one of the landmarks set.

Network Positioning System (NPS) [60] is another GNP variation that intends to reduce the traffic load experienced by the landmarks. NPS is a hierarchical coordinate system where all nodes can potentially be used as landmarks by the other nodes. For each node, the choice of the landmarks is done by a server. NPS still uses a set of permanent landmarks and these landmarks form the layer 0 of the architecture. The nodes using only permanent landmarks as landmarks form the layer 1 of the architecture. If the permanent landmarks are too heavily loaded or if they are unavailable, the server adds a layer 2 to the architecture and proposes nodes of the layer 1 as landmarks to the nodes of the layer 2. Finally, each node belonging to layer i randomly uses some nodes from layers below layer i as its landmarks.

Lim *et al.* [44] and Tang *et al.* [79] both proposed models based on *Lipschitz embedding*. The basic idea of the Lipschitz embedding is to use the measured distances as coordinates: the coordinate of node A is $(a_1, \dots, a_n)$ where a_i is the measured distance between A and the landmark i for $i = 1, \dots, n$. Since two nodes that are close to each other in the Internet have generally similar distances to the other nodes, two nearby nodes in the Internet will have similar coordinates in the Lipschitz embedding and will be considered as nearby nodes using the estimations. The two models begin by embedding the nodes in a n-dimentional space using their distances to the n landmarks (Lipschitz embedding). To reduce the dimentionality, the coordinates in the n-dimentional space are then projected in a d-dimentional space (with $d < n$) using a method called *principal component analysis (PCA)* [27].

Internet Distance Estimation Service (IDES) [54] proposes a model based on *matrix factorization*. The idea is to approximate a large matrix containing the distances between each pair of nodes belonging to the network by the product of two smaller matrices. To compute the small matrices from network measurements, IDES proposes two algorithms: *Singular Value Decomposition (SVD)* [34] and Non-Negative Matrix Factorization (NMF) [36]. For the measurements, IDES operates as a standard landmark-based approach. On one side, landmarks measure distances between them, and their "coordinates" (composed of an outgoing and an incoming vector in IDES) are computed using SVD or NMF applied on the full inter-landmarks delay matrix. On the other side, an ordinary node measures distances with the landmarks and computes its outgoing and incoming vectors using a least squares optimization algorithm applied using the measurement results and the outgoing and incoming vectors of the landmarks.

Phoenix [10] improves the approach proposed by IDES. Compared to IDES, Phoenix adopts a weighted model adjustment to achieve better prediction accuracy and ensures that the predicted distances are positive values (it was not the case with IDES and it could be a problem for some applications). Even if each node in Phoenix can potentially choose as reference point any node that has already computed its coordinate, Phoenix is not totally distributed. Indeed, during a first phase, Phoenix still considers the early-entered nodes as landmarks and a centralized algorithm (NMF) is used to compute the coordinates of these nodes.

Limitations

The main drawback of the landmark-based models is the usage of landmarks. A first major problem is that landmark failures and overloadings affect latencies. Consequently, the measurement results used to compute the coordinates become themselves inaccurate. Distributing the measurement load among all the nodes rather than on a small number of them would provide better results. Secondly, many studies [61, 54] have shown that the choice of the landmarks (their number and their positions) has an important impact on the accuracy of the estimations. Again, avoiding the use of the same set of reference points for all the nodes would reduce the impact of a bad choice of reference points. Even if some landmark-based approaches like Lighthouses or NPS tend to distribute the load on more nodes, they still require a dedicated infrastructure which is a point of failure.

These problems induce that landmark-based ICS are not attractive solutions for deployments at a large scale and we prefer fully distributed approaches.

3.2.2 Distributed coordinate systems

The distributed ICS extend the embedding concept proposed by the landmark-based solutions. Distributed approaches generalize the role of landmark to any node existing in the system (like NPS does but without the use of the centralized management server) and do not require any dedicated infrastructure.

State of the art

Practical Internet Coordinates (PIC) [14] is a variation of GNP that does not require explicitly designated landmarks. In PIC, any node that has already computed its coordinate can be used as landmark and a joining node has to use an active node discovery protocol to find its landmarks (there is no central server for this purpose). Moreover, PIC was the first ICS that introduced a security mechanism against malicious behaviours of nodes[1].

[1]With landmark-based approaches, it was possible to guarantee the reliability of the nodes belonging to the small set of landmarks. With a distributed approach, it is unrealistic to assume that no node in the network will behave maliciously. Consequently, a mechanism to protect the nodes against the behaviour of malicious nodes is necessary.

Big-Bang Simulation (BBS) [75] models the network as a set of particles where each particle is the image of a node in a geometric space. These particles move in that space under the effect of potential force field. Each pair of particles is pulled or repulsed by the field force induced between them depending on their embedding error. The main drawback of BBS is that its model is much more complicated than a model proposed a little bit later: Vivaldi.

Vivaldi [15] is probably the most successful ICS that has been proposed so far. It is fully decentralized (it does not require any fixed network infrastructure, it makes no distinction between nodes and each node can compute its own coordinate independently), its model is quite simple and it has been widely deployed and tested. Like BBS, Vivaldi models the nodes of the network as a set of elements that interact in a geometric space. But, where BBS uses a complicated model based on force fields, Vivaldi models the forces between the elements by using simple springs. In Vivaldi, each node chooses a few reference points called its *neighbors*. The idea is that a node A is connected to each neighbor B by a spring which has a rest length corresponding to the measured distance between A and B (*i.e.* $d(A, B)$) and an actual length coresponding to the estimated distance in the coordinate space between A and B (*i.e.* $\hat{d}(A, B)$). The goal of a node is to have a coordinate that minimizes the potential energy of the springs attached to it, *i.e.* each node adapts its coordinate to have, for each spring, an actual length which is as near as possible to the rest length. Thus, if $\hat{d}(A, B)$ is smaller than $d(A, B)$, the spring between the node A and its neighbor B pushes A away from B to increase its actual length. On the contrary, if $\hat{d}(A, B)$ is bigger than $d(A, B)$, the spring pulls the node towards its neighbor to reduce its actual length. More details about Vivaldi's mechanism will be given in section 3.3.

Decentralized Matrix Factorization (DMF) [42] is, like IDES and Phoenix, a mechanism based on matrix factorization. Like in Phoenix, a node in DMF can choose any node in the network as reference point. But, unlike Phoenix, DMF get rid of the first phase where a complete delay matrix between the early-entering nodes and a centralized algorithm were required to compute the coordinates (incoming and outgoing vectors) of these nodes: DMF is a fully distributed approach. In a first version of DMF [42], Liao *et al.* solved this problem by initializing the coordinates of the nodes with random numbers. Then, they use a least squares optimization algorithm to update the coordinates of the nodes. They observed that DMF is insensitive to the random initialization and that the coordinates of nodes converge progressively to good values.

Recently, Liao *et al.* proposed a new version of DMF called *Decentralized Matrix Factorization by Stochastic Gradient Descent (DMFSGD)* [41, 40]. With this version, they formulate the problem of network distance prediction as a matrix completion problem: the delay matrix contains the results of the measurements done between the nodes and their reference points, but lots of delays are not measured and must be inferred from the measured ones. Matrix completion is only possible if matrix entries are largely correlated and this is the case for network distances[2]. The matrix completion process is done

[2]Intuitively, for two near nodes, the distances to the other nodes are similar: for a same destination, these nodes use the same Internet paths. Consequently, the lines corresponding to these nodes in the delay matrix are correlated.

using a fully decentralized algorithm based on Stochastic Gradient Descent (SGD) [2]. More details about DMFSGD will be given in chapter 7.

Discussion

Distributed approaches are generaly more attractive than landmark-based approaches because they do not rely on a dedicated infrastructure. It has mainly two consequences. The first one is that they are more robust: in case of outage or attack, a dedicated infrastructure is a point of failure and can prevent the whole system to work. The second one is that they are more scalable: with a distributed approach there is no problem of overloading of the landmarks.

On the other side, distributed approaches are criticized for being more vulnerable to security threats. Indeed, in a landmark-based approach, it is possible to ensure that all the landmarks are reliable nodes and to ensure that each node uses reliable reference points to compute its coordinates. In a distributed approach, it is difficult to ensure that each node in the system is reliable. Consequently, some nodes can compute bad coordinates caused by malicious nodes used as reference points.

Distributed approaches are also criticized for having worse prediction accuracy than landmark-based approaches. However, implementations and deployements in the Internet have shown that P2P applications and overlays relying on the notion of network proximity can benefit from the estimations provided by distributed ICS. For example, using the Azureus [80] BitTorrent network as a testbed (with about one million nodes), Ledlie *et al.* [35] have shown that, even if the coordinates produced by Vivaldi give sometimes large estimation errors when Vivaldi is used in the Internet, these coordinates achieve the major goal they were designed for: deliver a reasonably accurate position for each node, allowing quite reliable approximation of nodes proximity.

3.2.3 Choosing one suitable ICS to reach our objective

Since our main objective is to improve RON's scalability, we need an ICS that is suitable for a large scale deployment. So, we naturally tend to choose a distributed ICS. This choice is reinforced by the fact that RON is, by definition, a distributed system (each node is able to find an alternative path itself when it is necessary). So, using a centralized measurement mechanism has no sense: this would only make the system more vulnerable to outages and attacks. Among distributed ICS, Vivaldi seems to be a good choice. Compared to other distributed approaches, choosing Vivaldi gives many advantages:

- Vivaldi has already been widely studied and tested (and not only in specific testbeds like Planetlab). So, we know that it is able to provide quite accurate estimations, even in networks composed of about one million nodes.
- Multiple improvements have been proposed for Vivaldi (see section 3.5). These improvements can give better (or worse) results for our intended usage of the coordinates.
- A simulator developed by the Vivaldi's conceptors is publicly available and can be used for experiments (see section 3.3.2).

To benefit from all these advantages, we have choosen Vivaldi as estimation mechanism to improve RON's scalability.

However, the recent decentralized approach based on matrix factorization seems also interesting. Indeed, following its conceptors, DMFSGD provides more accurate estimations than Vivaldi and DMFSGD could be a better choice than Vivaldi to reach our goal. Since that system was not available at the beginning of our work, it was not our primary choice. Nevertheless, we will investigate the results obtained with that estimation mechanism and compare these results to those obtained with Vivaldi in chapter 7.

3.3 Vivaldi

Vivaldi [15] is a popular fully distributed ICS that does not require any particular network infrastructure. In Vivaldi, each node computes its own coordinate by doing measurements with a few other nodes called its *neighbors*. If m denotes the number of neighbors chosen by each node (typically $m = 32$), the measurement cost of Vivaldi in a network containing n nodes is $O(n \times m)$. Since m is a very small value compared to n, this is better than the measurement cost $O(n^2)$ obtained if all paths must be measured (like in RON).

3.3.1 Algorithm

As previously explained, Vivaldi simulates a spring between each pair of nodes (A, B) where B is a neighbor of A. The rest length of the spring placed between the node A and its neighbor B is equal to the measured RTT ($RTT(A, B)$) for the path AB and its current length is equal to the estimated RTT $EST(A, B)$ for the path AB (computed using the coordinates of the nodes). The potential energy of the spring is equal to zero when its actual length is equal to its rest length. In other words, the potential energy of a spring is equal to zero when the estimation error of the corresponding path is equal to zero. In such conditions, each node adjusts its coordinate to minimize the potential energy of the springs placed between it and its neighbors in order to minimize the estimation errors in the network. An identical Vivaldi procedure runs on every node.

Basic principle

Let $\vec{F}(A, B)$ be the force applied on node A by the spring placed between node A and its neighbor B. Following Hooke's law, we have

$$\vec{F}(A, B) = (RTT(A, B) - EST(A, B)) \times u(\vec{x}_A - \vec{x}_B)$$

where $\vec{x}_A$ is A's coordinate, $\vec{x}_B$ is B's coordinate, $EST(A, B) = \|\vec{x}_A - \vec{x}_B\|$ is the estimated RTT for the path AB and $u(\vec{x}_A - \vec{x}_B)$ is a unit-length vector that gives the direction of the force. The total force experienced by A is:

$$\vec{F}(A) = \sum_{B \in N(A)} \vec{F}(A, B)$$

where $N(A)$ is the set containing A's neighbors. To simulate the spring network evolution, each node starts with a coordinate located at the origin of the space[3] and the mechanism considers small intervals of time. At each interval, the mechanism computes the force experienced by each node and moves each node in the space of a small distance depending of the force experienced by the node. At the end of a time interval of size δ, the coordinate of a node A is modified using the following formula:

$$\vec{x}_A = \vec{x}_A + (\vec{F}(A) \times \delta)$$

This mechanism converges progressively towards a situation that minimizes the potential energy of the springs (*i.e.* the estimation errors in the network).

Finding an appropriate value for δ is important in the design of Vivaldi. If that value is too big, the nodes will move on long distances at each step and it could finally be impossible for them to find a good position in the geometric space. On the other hand, if that value is too small, the system will take a long time to converge.

Simple algorithm

Based on this principle, the designers of Vivaldi proposed a first simple algorithm. In their approach, a node updates its coordinate after each measurement with one of its neighbors. When node A measures the RTT with its neighbor B (and learns B's current coordinate during the process), A updates its coordinate using the following rule:

$$\vec{x}_A = \vec{x}_A + \delta \times (RTT(A,B) - EST(A,B)) \times u(\vec{x}_A - \vec{x}_B)$$

where δ is a constant timestep. Algorithm 1 gives an example of a procedure called when a measurement is done between a node and one of its neighbors. The intensity of the force (*i.e.* the absolute estimation error (AEE) of the sample) is computed in line 4, the direction of the force is computed in line 5, the force vector is computed in line 6 and the local node's coordinate is updated in line 7.

Algorithm 1 Simple_Vivaldi (rtt, $\vec{x}_{remote}$)

Require: A RTT measurement has been done between the node and one of its neighbors.

```
1: {$\vec{x}_{local}$ is the coordinate of the local node}
2: {$\vec{x}_{remote}$ is the coordinate of the neighbor}
3: {$rtt$ is the measured RTT}
4: $int = rtt - \|\vec{x}_{local} - \vec{x}_{remote}\|$
5: $dir = u(\vec{x}_{local} - \vec{x}_{remote})$
6: $f = dir \times int$
7: $\vec{x}_{local} = \vec{x}_{local} + \delta \times f$
```

The main drawback of that simple algorithm is that it is biased towards more recent samples. For example, if a node has already a good position in the geometric space

[3]Since all the nodes start at the same location, Vivaldi must find a way to separate them. For this purpose, Vivaldi defines $u(0)$ as a unit-length vector in a randomly chosen direction.

with regard to all its neighbors excepting one of them and if it does a measurement with that problematic neighbor, then the algorithm moves the node in the geometric space with respect to the estimation error of the last sample without considering the previous samples. To circumvent this problem, each node could maintain a list of every samples received so far and consider all these samples during the coordinate update process. However, since the nodes are constantly updating their coordinates, old samples can be outdated and maintaining such a list is not scalable. The solution proposed by Vivaldi is to add a notion of *confidence* in the coordinates and to use an adaptive timestep instead of a constant one.

Timestep's value

Vivaldi uses an adaptive value for the timestep δ. The timestep is computed during each coordinate update and it depends of two values:

The node's error (*local error*): If the confidence of the node in its coordinate is low, the node can make important moves in the geometric space in order to reach quickly a more accurate position. In other words, if the local error is high, then the coordinate update must use a big value for the timestep. Reversely, if the confidence of the node in its coordinate is high, it has only to refine its coordinate: its global position in the geometric space considering the positions of its neighbors is already quite good. In such situation, the node must only do small moves in the geometric space. In other words, if the local error is low, then the timestep must be a small value.

The neighbor's error (*remote error*): If the neighbor used for the coordinate update has a low confidence in its coordinate, the node must moderate the impact of the corresponding sample (in particular if the node has a high confidence in its current coordinate). Consequently, if the remote error is high and the local error is low, then the timestep must be a small value.

To take all these cases into account during the coordinate update process, Vivaldi defines the *weight* of the sample as the ratio $w = \textit{local error}/(\textit{local error}+\textit{remote error})$. The adaptive timestep is computed using the following formula where $cc < 1$ is a constant positive value:

$$\delta = cc \times \frac{\textit{local error}}{\textit{local error} + \textit{remote error}}$$

Estimating the local error

To compute the adaptive timestep, the node requires an estimation of its local error. Vivaldi updates the local error of a node after each measurement with one of its neighbors. Using the measured RTT and the neighbor's coordinate, the node can compute the *sample error* which is the estimation error for the path between him and its neighbor. For the node A and its neighbor B, we have:

$$e_s = |REE(A,B)| = \frac{|RTT(A,B) - EST(A,B)|}{RTT(A,B)}$$

If the sample error is high (*resp.* low), the node can reduce (*resp.* increase) its confidence in its coordinate, *i.e.* it can increase (*resp.* reduce) its local error. To do this, each node maintains a weighted average of all the sample errors computed so far and the result of the average is the local error of the node. Obviously, the impact of the last sample error on the average must depend of the weight of the sample. For example, for a node with a low local error, if a sample reveals a high estimation error with a neighbor which has a low confidence in its coordinate, then the impact of the high sample error on the average must be moderated: since the position of the node is accurate with regard to the positions of the other neighbors, the high estimation error observed for this sample is probably caused by a bad current position of the probed neighbor. To update the node's local error, Vivaldi uses the following formula where $ce < 1$ is a positive constant:

$$\textit{local error} = e_s \times (ce \times w) + \textit{local error} \times (1 - (ce \times w))$$

Vivaldi's algorithm

Algorithm 2 is the Vivaldi algorithm. A node calls this procedure each time it does a measurement with one of its neighbors. The weight of the sample is computed in line 6, the error of the sample is computed in line 8, the local error is updated in line 9, the adaptive δ is computed in line 10, the force vector is computed in lines 11 to 13 (like in the simple version of the algorithm) and the local coordinate is updated in line 14 (using the adaptive timestep).

Algorithm 2 Vivaldi (rtt, $\vec{x}_{remote}$, e_{remote})

```
Require: A RTT measurement has been done between the local node and one of its
    neighbors.
 1: {$\vec{x}_{local}$ is the coordinate of the local node}
 2: {$\vec{x}_{remote}$ is the coordinate of the neighbor}
 3: {$e_{local}$ is the node's local error}
 4: {$e_{remote}$ is the neighbor's local error (the remote error)}
 5: {$rtt$ is the measured RTT}
 6: $w = e_{local}/(e_{local} + e_{remote})$
 7: $est = \|\vec{x}_{local} - \vec{x}_{remote}\|$
 8: $e_s = |rtt - est|/rtt$
 9: $e_{local} = e_s \times (ce \times w) + e_{local} \times (1 - (ce \times w))$
10: $\delta = cc \times w$
11: $int = rtt - est$
12: $dir = u(\vec{x}_{local} - \vec{x}_{remote})$
13: $f = dir \times int$
14: $\vec{x}_{local} = \vec{x}_{local} + \delta \times f$
```

3.3.2 P2PSim simulator

The P2PSim simulator [64] is a discrete-event simulator which comes with an implementation of the Vivaldi system. We have used that simulator to conduct our experiments. The implementation of Vivaldi proposed by this simulator has been created by F. Dabek and R. Morris that are two of the authors of the Vivaldi reference paper (*i.e.* [15]). However, we have noticed two small differences between that implementation and the theoretical algorithm presented in section 3.3.1.

The first difference between the implementation in the simulator and algorithm 2 appears during the update of the local error: in line 6 of the algorithm, instead of computing w, the implementation computes w' with

$$w' = \frac{e_{local}^2}{e_{local}^2 + e_{remote}^2}$$

In other words, the implementation uses squared errors to compute the weight of the sample instead of using the errors. The consequence of that modification is only that the weight of the sample will be higher when it must be high and it will be lower when it must be low (see example 1).

Example 1. *Let's first consider the case where the local error is small and the remote error is big. For example,* $e_{local} = 0.1$ *and* $e_{remote} = 0.8$*. In such situation, the weight of the sample must be low: since the node is confident in its actual coordinate while the neighbor is not, the reference point is not reliable and the impact of the sample on the node's coordinate must be small. We have*

$$w = \frac{0.1}{0.1 + 0.8} = 0.111 \qquad w' = \frac{0.1^2}{0.1^2 + 0.8^2} = \frac{0.01}{0.01 + 0.64} = 0.015$$

As intended, the value of w *is small and the value of* w' *is even smaller. Now, consider the case where the local error big and the remote error is small. For example,* $e_{local} = 0.8$ *and* $e_{remote} = 0.1$*. In such situation, the weight of the sample must be high: since the node is not confident in its actual coordinate while the neighbor is, the reference point is reliable and the impact of the sample on the node's coordinate must be important. We have*

$$w = \frac{0.8}{0.8 + 0.1} = 0.889 \qquad w' = \frac{0.8^2}{0.8^2 + 0.1^2} = \frac{0.64}{0.64 + 0.01} = 0.985$$

As intended, the value of w *is big and the value of* w' *is bigger.*

The second difference between the implementation in the simulator and algorithm 2 appears while δ is computed. Like in algorithm 2, the implementation computes $\delta = cc \times w$ (to compute δ, the implementation uses w and not w') but the difference is that w is computed by the implementation after the update of the local error (*i.e.* between line 9 and line 10 in algorithm 2). Consequently, the variable e_{local} used to compute w has the updated error as value in the implementation while it has the error before the update as value in algorithm 2. Using the updated error instead of the error before the update can be justified. Indeed, on one side, if the local error is reduced (resp. increased) during the update process, then the weight computed by the implementation will be smaller (resp. bigger) than the weight computed in algorithm 2. On the other hand, if the local error is reduced during the update process then, the estimation error of the path between the node and its neighbor is already quite small. Consequently, having a smaller value for the weight and doing only a small move in the metric space to keep approximatively the same estimation is a good behavior. Reversely, if the local error is increased during the update process then, the estimation error of the path between the node and its neighbor is important. In such situation, having a bigger value for the weight and doing a large move in the metric space to improve the estimation is a good behavior.

As conclusion, even if there are differences between the theoretical algorithm and the implementation in the simulator, the implementation has the same general philosophy as the theoretical algorithm. So, these differences are acceptable. But the question is, are they necessary? It is difficult to answer this question. To our knowledge, there is no paper on that subject and there is no comments in the implementation to explain those modifications. We have done some small experiments and we have not observed major differences between the results obtained with the P2PSim implementation of Vivaldi and the results obtained with an exact implementation of the Vivaldi algorithm. However, that does not mean that there are no cases where these modifications are necessary to obtain good estimations. Since the creators of Vivaldi (who have implemented it in the P2PSim simulator) know their algorithm better than us, they had probably a good reason to do these modifications and we will take their implementation as it is for our own experiments.

3.3.3 The embedding space

The Vivaldi algorithm uses the concept of coordinate without defining exactly what is a coordinate. Actually, Vivaldi works with any coordinate system that supports the operations used by the algorithm: coordinate subtraction, vector norm and scalar multiplication operations.

Definition 9. *In geometry, a* coordinate system *is a system which uses one or more numbers (or coordinates) to uniquely determine the position of a point.*

To allow the definition of the operations required by Vivaldi, the coordinate system must be defined on a metric space.

Definition 10. *A* metric space *is an ordered pair* (M, d) *where* M *is a set and* d *is a metric on* M.

Definition 11. *A* metric d *on a set* M *is a function* $d : M \times M \to \mathbb{R}$ *such that for any* x, y, $z \in M$, *the following holds:*

1. *Non-negativity:* $d(x, y) \geq 0$
2. *Identity of indiscernibles:* $d(x, y) = 0$ *if and only if* $x = y$
3. *Symmetry:* $d(x, y) = d(y, x)$
4. *Triangle inequality:* $d(x, z) \leq d(x, y) + d(y, z)$

Type of spaces

There exist many types of metric spaces that can be used with Vivaldi. Since Vivaldi simulates a real-world spring system (*i.e.* a system working in a three dimensional Euclidean space), a natural choice is to use a d-dimensional Euclidean space for Vivaldi. The definitions in the Euclidean space of the operations used by the algorithm are the following:

$$\begin{aligned}(x_1, \cdots, x_d) - (y_1, \cdots, y_d) &= (x_1 - y_1, \cdots, x_d - y_d) \\ \|(x_1, \cdots, x_d)\| &= \sqrt{x_1^2 + \cdots + x_d^2} \\ \alpha \times (x_1, \cdots, x_d) &= (\alpha x_1, \cdots, \alpha x_d)\end{aligned}$$

Some people tried to use Vivaldi with other types of coordinates hoping to obtain a better prediction accuracy. Since the distances that Vivaldi tries to embed in the metric space are obtained from paths that are on the surface of the Earth (*i.e.* a sphere) another natural choice is to use Vivaldi with spherical coordinates. Dabek *et al.* [15] observed that the estimation errors obtained with Vivaldi are bigger with spherical coordinates than with simple 2D-Euclidean coordinates. Using their measurement results, Ledlie *et al.* [35] showed that the Internet is a "flat" world and not a sphere because most of the traffic between Europe and Asia flows through North America. Consequently, putting European nodes on one side of a Euclidean space, Asian nodes on the opposite side and American nodes in the middle gives better estimations than putting the nodes on a sphere.

It has also been proposed to use Vivaldi with a Euclidean space augmented with a height. In such space, each node has a coordinate $[\vec{x}, x_h]$ where $\vec{x}$ is the coordinate of the node x in a Euclidean space and x_h is the height of the node x. A packet sent from one node to another must travel the source node's height, then travel in the Euclidean space and, finally, travel the destination node's height. Intuitively, the height of a node represents the latency of the node's access link to the Internet and the distance in the Euclidean space represents the latency in the Internet between the source's access point and the destination's access point. The defintions of the operations used by Vivaldi are the following:

$$\begin{aligned}[\vec{x}, x_h] - [\vec{y}, y_h] &= [(\vec{x} - \vec{y}), x_h + y_h] \\ \|[\vec{x}, x_h]\| &= \|\vec{x}\| + x_h \\ \alpha \times [\vec{x}, x_h] &= [\alpha\vec{x}, \alpha x_h]\end{aligned}$$

Dabek *et al.* [15] observed that Vivaldi with a two dimensional Euclidean space augmented with a height provides slightly more accurate estimations than Vivaldi with a three dimensional Euclidean space. However, with that type of space, Vivaldi tends to over-estimate small RTTs. This happens mainly for two reasons. The first one is linked to the intuitive idea itself. Indeed, to travel from one node to another, it is not always necessary to travel through the access links of these nodes (*e.g.* if the two nodes are in the same LAN). However, in such situation, the estimation considers the height of the two nodes and over-estimates the RTT. The second reason is that the height must be a strictly positive value but can become negative or equal to zero with the Vivaldi algorithm: typically, if the node is too far from its neighbor, the intensity of the force vector (line 11 in algorithm 2) is negative and can lead to a negative value for the height of the local node after the coordinate update. If the height of a node is negative at the end of a coordinate update, node's height must be set to an arbitrary positive value to avoid future problems. Even if that positive value is small, the operation is opposed to the behavior of the algorithm. Indeed, in such situation, the algorithm modifies the coordinate of the node to make it closer to its neighbor. Resetting manualy the height to a positive value when it is negative or equal to zero means putting the node farther from its neighbor (and even possibly farther than it was before the coordinate update). Consequently, Vivaldi has difficulties to estimate small RTTs when it uses a Euclidean space augmented with a height vector.

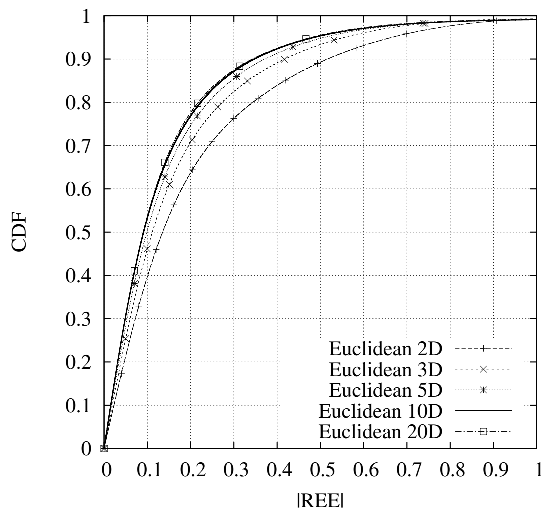
Shavitt and Tankel [76] tried to embed RTTs in a hyperbolic space and obtained, with BBS, a better prediction accuracy than when they used a Euclidean space. In their paper Dabek *et al.* [15] cite this possibility as a future work and they said that their model consisting in an Euclidean space augmented with a height is already a rough approximation of an hyperbolic space. To our knowledge, they have never written a paper presenting such results. Later, Lumezanu and Spring [52] tried to use Vivaldi with hyperbolic coordinates. They compared that model to the Euclidean model and observed that the hyperbolic model performs better in some situations while the Euclidean model performs better in other situations.

Choice of a Euclidean space

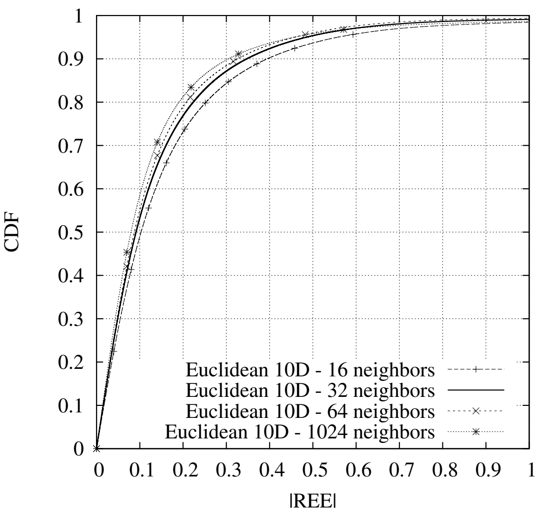
Our choice is to use a pure Euclidean space for our experimentations. Since spherical coordinates provide worse estimations, we can ignore that possibility. Since our goal is to use the estimations to find routing shortcuts, we intend to use the estimations to find the smaller RTTs. Consequently, having a coordinate system that tends to over-estimate small RTTs is not interesting and we do not choose the Euclidean augmented with a height model. So, we have to choose between the Euclidean model and the hyperbolic model. Since these models are quite equivalent in terms of prediction accuracy, we finally prefer the Euclidean model for its simplicity.

We will use Vivaldi with a d-dimensional Euclidean space but we still need to choose a value for d. Since Vivaldi simulates springs working in a 3-dimensional space, choosing a 3-dimensional Euclidean space would be natural. The principal component analysis conducted by Tang and Crovella [79] on small datasets also suggests that a 3-dimensional Euclidean space is sufficient but, using the large amount of data collected through Azureus, Ledlie *et al.* [35] found that four or five dimensions is more appropriate for an Internet-scale system. The goal of these studies was to find a tradeoff between prediction accuracy and the communication overhead required by the prediction mechanism. Our goal is a little bit different. Even if our initial objective is to reduce the communication and measurement overheads of RON's approach, we mainly want to demonstrate that this is feasible using the prediction provided by a coordinate system. Consequently, we want as much prediction accuracy as possible and we allow the use of a little bit more dimensions than what is recommended by these studies. Moreover, as long as $d \lll n$, the communication overhead of an estimation based approach will be smaller than RON's communication overhead (which is $O(n^3)$). Indeed, since n nodes will have to send their coordinate to n other nodes, the number of messages is $O(n^2)$. If the size of a coordinate is $O(d)$, the total communication overhead with an estimation based approach is $O(n^2 \times d)$.

To choose a value for d, we simulated Vivaldi in a d-dimentional Euclidean space with different values for d. For these simulations, we used the RTT matrix provided with the P2PSim simulator which contains the RTTs measured in the Internet between 1740 nodes (more information about that matrix can be found in section 3.4). Except the value of d, all the parameters are identical for each simulation. In particular, each node uses the same set of 32 neighbors. At the end of each simulation, we computed the estimation of each RTT using the coordinates and we computed the relative estimation error (REE - see definition 8 page 25) for each RTT. Figure 3.2(a) gives the CDFs (Cumulative Distribution Functions) of the REEs obtained with different values of d.



(a) Varying the number of dimensions



(b) Varying the number of neighbors

Figure 3.2: **Impact of parameters variations on Vivaldi's estimation errors.** The left figure represents the CDFs of the REEs obtained by running Vivaldi in a d-dimensional Euclidean space with 32 neighbors for different values of d. The right figure represents the CDFs of the REEs obtained by running Vivaldi in a 10-dimensional Euclidean space with m neighbors for different values of m.

Figure 3.2(a) confirms the results obtained by Ledlie *et al.*: using a 5-dimensional space seems sufficient to ensure a good preduction accuracy. For example, with a 5-dimensional space, figure 3.2(a) shows that there are 75% of the RTTs which have a REE smaller or equal to 0.2. As comparison, in a 3-dimensional (resp. 2-dimensional) space, there are only 70% (resp. 65%) of the RTTs which have a REE smaller or equal to 0.2. It is still possible to get a little accuracy improvement by using a 10-dimensional Euclidean space. In such space, there are 77% of the RTTs which have a REE smaller or equal to 0.2. However, going above ten dimensions is clearly useless: for example, in figure 3.2(a), we see that the CDF obtained by running Vivaldi in a 20-dimensional Euclidean space is almost the same than the CDF obtained in a 10-dimensional Euclidean space.

In order to avoid as much as possible prediction inaccuracies caused by the choice of the space, we choose to conduct all our experiments in a 10-dimensional Euclidean space. However, we keep in mind that this is a "security" choice and that we should probably obtain similar results by using only a 5-dimensional Euclidean space (*i.e.* with a communication overhead divided by two).

3.3.4 Neighbors

Another parameter of the Vivaldi algorithm that has an large impact on the estimation accuracy is the neighbors used by the nodes to compute their coordinates: the number of neighbors used by the nodes and the way they are selected are important.

Neighbors selection strategy

In [14], Costa *et al.* have shown that the way the neighbors are chosen has a large impact on the estimation accuracy. They implemented three different strategies to choose the m neighbors of a given node A:

Random: Select the m neighbors randomly in the set of nodes participating in the ICS.

Closest: Select the m nodes participating in the ICS that are the closest to A.

Hybrid: Select some neighbors randomly and others as in the closest strategy.

Their experiments have shown that the closest strategy leads to coordinates that give good estimations for short distances but also large estimation errors for long distances. Such neighbors selection strategy is interesting for applications that require essentially accurate estimations for short distances: finding the closest server, finding the nearest node in a P2P network, *etc.* This is not a good choice in our case because we require also quite accurate estimations for long distances.

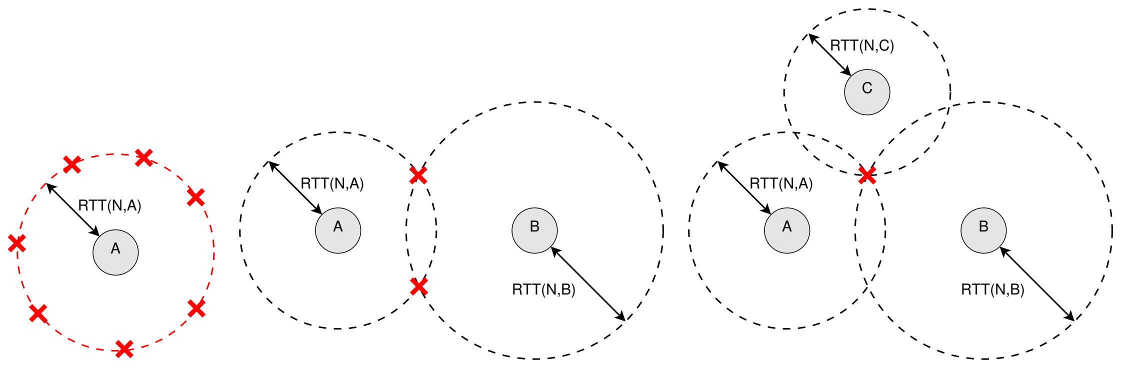
The random neighbors selection strategy, is the opposite. With that strategy, there is only a small probability to have a significant number of A's neighbors that are close to A in the topology and A computes its coordinate essentially with neighbors that are far from it. It leads to a good global position in the space for node A (*i.e.* small estimation errors for long distances) but it leads also to a rough position with respect to the nodes that are close to it (*i.e.* large estimation errors for short distances).

The hybrid strategy is the best choice. It benefits from the advantages of the closest strategy (good estimations of short distances) and the advantages of the random strategy (good estimations of long distances). It remains to choose the percentage of neighbors that are respectively chosen as in the random strategy and as in the closest strategy. Dabek *et al.* [15] have shown that even when only 5% of the neighbors are distant nodes, it is possible to avoid the bad global positioning problem experienced with the closest strategy. However, with only a small percentage of distant nodes in the neighbors sets, Vivaldi converges slowly to accurate coordinates. When half of the neighbors are distant/random nodes and half of the neighbors are the closest nodes, Vivaldi converges quickly towards coordinates that allow good estimations for long and short distances.

In conclusion, for our experiments, we will use the hybrid neighbors selection strategy with half of the neighbors that are the closest nodes and half of the neighbors that are distant/random nodes.

Number of neighbors

We will now choose the number of neighbors m that will be used by each node. There is one constraint on m: if Vivaldi runs in a d-dimensional Euclidean space, each node must use at least $d + 1$ neighbors to compute its coordinate. If m does not meet that condition, the nodes have not enough information to find their optimal position in the space. The problem is illustrated in figure 3.3 for a 2-dimensional Euclidean space. Figure 3.3(a) illustrates the situation with one neighbor and figure 3.3(b) illustrates the situation with two neighbors. In these situations, there exist many optimal coordinates with respect to the coordinates of the node's neighbors but it is impossible to know which of these coordinates is optimal with respect to the coordinates of all the other nodes. If the node uses at least three neighbors to compute its coordinate (figure 3.3(c)), there is only one optimal coordinate with respect to the coordinates of its neighbors (and, with respect to the coordinates of all the other nodes). Since we work with a 10-dimensional space, we nead at least 11 neighbors per node.



(a) One neighbor (b) Two neighbors (c) Three neighbors

Figure 3.3: **Lower bound on the number of neighbors.** In a 2-dimensional space, if a node N uses only one neighbor A to compute its coordinate (figure 3.3(a)), it can select any coordinate located at a distance $RTT(N, A)$ of A's coordinate. If it uses two neighbors (figure 3.3(b)), it has still two optimal coordinates with respect to the coordinates of its neighbors. If it uses (at least) three neighbors (figure 3.3(c)), the node N has only one optimal coordinate with respect to the coordinates of its neighbors.

Using a 3-dimensional space and a hybrid neighbors selection strategy, Dabek *et al.* [15] observed that Vivaldi's prediction accuracy increases rapidly until about 32 neighbors and does not improve much with more neighbors. Since we intend to use more than three dimensions, we repeated their experiments using a 10-dimensional space and a hybrid neighbors selection strategy. The CDFs of the REEs obtained using different numbers of neighbors are given in figure 3.2(b) page 38. In this figure, we see that 32 neighbors is also a good choice with a 10-dimensional space. It is still possible to obtain a small improvement by using 64 neighbors but it multiplies the measurement costs by two. It is clear that going above 64 neighbors is useless. For example, the figure shows that using a very big number of neighbors (1024 on the 1739 nodes available) does not improve a lot the prediction accuracy compared to simulations with 32 or 64 neighbors. Regarding those results, we choose to conduct our experiments using 32 neighbors.

3.3.5 Other Vivaldi parameters

In order to run the Vivaldi algorithm, it is still necessary to choose values for the parameters cc and ce. In their paper, Dabek *et al.* [15] stated that a cc value of 0.25 yields both quick convergence to accurate coordinates and low coordinate oscillations. A few years later, Elser *et al.* [19] stated that the optimal value for the constant cc is 0.005. However, considering their graphs, using $cc = 0.25$ seems a quite good choice and, even if using smaller values for cc improves the stability of the coordinates, it also increases significantly the convergence time. Since we intend to use Vivaldi to improve RON's approach, having a small convergence time is crucial to have a system reacting quickly to network topology changes. Consequently, for our experiments, we will use $cc = 0.25$.

For the ce value, Dabek *et al.* do not give any value in [15]. In the P2PSim simulator, they used $ce = 0.05$. In [19], Elser *et al.* observed that ce has a very small impact on the prediction accuracy and the stability of the coordinates. Following these observations, we will use the default ce value proposed by the simulator for our experiments.

3.4 Data sets used in simulations

Remember that our main goal is to demonstrate that it is possible to improve the routes in the Internet (like RON does) in a scalable manner by using an estimation-based approach (instead of the RON's measurement-based approach). We intend to evaluate the efficiency of the estimation-based approach by using the P2PSim simulator which provides an implementation of the Vivaldi algorithm. To run simulations, we need a description of a topology. Since Vivaldi estimates RTTs, that description must provide the RTTs between the nodes of the topology. For this purpose, a simple $n \times n$ square matrix M such that $M[i][j] = RTT(node_i, node_j)$ is sufficient to describe a topology composed of n nodes. For a given simulation, the simulator simply uses as inputs the elements $M[i][j]$ of M such that $node_j$ is a neighbor of $node_i$ (and it ignores the other elements of M since its goal is to estimate them).

So, we need RTT matrices as inputs for our simulations. But we do not want any RTT matrix. Since our goal is to demonstrate that an estimation-based approach can improve routing efficiency at Internet-scale, we need large matrices obtained by doing real measurements in the Internet. We used the *P2PSim* data (1740 nodes) [64] and *Meridian* data (2500 nodes) [83] to model Internet latency. These data sets have been obtained following the *King* [23] measurement technique. King is a technique (similar to *ping*) that estimates the latency between arbitrary end hosts by using recursive DNS queries. We will also use a third RTT matrix obtained by doing measurements between 180 nodes in the Planetlab [69] research network. Since the *Planetlab* data set is not representative of the situation in the Internet[4] and describes only a small topology compared to the other two data sets, we will essentially focus on the P2PSim and the Meridian data sets for our experiments and use the Planetlab data set as comparison when it is necessary.

Even if P2PSim and Meridian have been both obtained by doing measurements in the Internet, there are large differences between the characteristics of these data sets. Meridian describes a topology containing very long RTTs (with a maximum of 3 seconds) but where most of the RTTs are small (90% of the RTTs are smaller than $150\,ms$). The topology described by the P2PSim data set contains less "anomalies" (the biggest RTT is $800\,ms$) and the RTTs are more distributed among small and middle values (85% of the RTTs are smaller than $300\,ms$). For comparison, even if Planetlab nodes are all over the world, the biggest RTT in that topology is $300\,ms$ and 96% of the RTTs are smaller than $200\,ms$ (see figure 3.4).

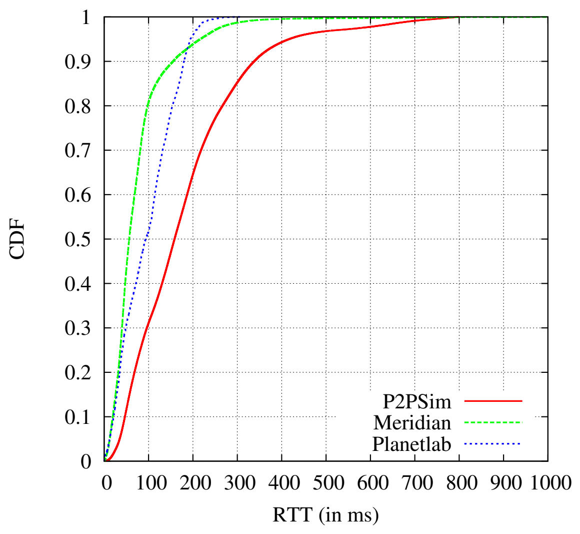


Figure 3.4: **Distribution of the RTTs in the topologies.** That figure gives the CDF of the RTTs (in milliseconds) for the three topologies we will use to conduct our experiments.

3.5 Vivaldi's improvements

Over the years, many improvements have been proposed for Vivaldi hoping to obtain more accurate coordinates. We have already discussed some of these improvements (*e.g.*

[4]The Planetlab research network is composed of nodes mainly located in universities or large companies. Lots of these institutions are interconnected through research dedicated networks like GEANT rather than being interconnected through the public Internet. Consequently, routes between these nodes are less dependent of BGP economic rules and can generally be more optimal than corresponding routes in the public Internet.

working with particular coordinate spaces) and we will not come back to these small modifications of Vivaldi. However, some people proposed deeper modifications of Vivaldi. Among, these modifications, we can cite the algorithm modifications and the attempts to give a structure to Vivaldi.

3.5.1 Algorithm modifications

An example of modification of Vivaldi's algorithm has been proposed by de Launois *et al.* [16]. With SVivaldi, they propose a method for stabilizing coordinates: they introduce a loss factor in the algorithm which represents the energy lost at each oscillation of springs and allows the springs to progressively rest in a local minimum. The timestep is computed the following way:

$$\delta = cc \times w \times (1 - loss)$$

The $loss$ variable starts with a value equal to 0 (no impact on the timestep computation) and grows progressively. While this factor mitigates coordinate oscillations in a fixed network and allows more accurate estimations, it prevents the algorithm from adapting to changes in the topology[5]. This is a real drawback for us: we need a coordinate system reacting as fast as possible to network changes and not only to big network changes.

Another modification of the algorithm has been proposed by Wang *et al.* [82]. In their paper, they state that simply applying a non-linear transformation to the RTTs before trying to embed them in the metric space leads to more accurate estimations. We will investigate that idea in chapter 6.

3.5.2 Structured Vivaldi

We have investigated the possibility to define a hierarchical structure among the nodes in order to improve the prediction accuracy [29]. Chen *et al.* have done the same and proposed Pharos [11]. These systems are based on a clustering of the nodes to propose a two-layer model for Vivaldi. Each node maintains two coordinates: it computes a local coordinate using neighbors in its cluster (the lower level of the hierarchical structure) and it computes a global coordinate using neighbors that are inside and outside its cluster (the upper level of the hierarchical structure). The global coordinates are the same as the coordinates computed by a classical Vivaldi and are used to compute estimations between nodes that are not belonging to the same cluster. The local coordinates are used to compute estimations between nodes belonging to the same cluster. The studies have shown that intra-cluster estimations are more accurate using the local coordinates than using the global coordinates. Recently, an improved version of Pharos called Tarentula [12] has been proposed. Tarentula uses a more sophisticated clustering technique than the simple two-layer model in order to improve the prediction accuracy. The hierarchical approach for Vivaldi will be investigated in chapter 8.

[5]de Launois *et al.* state that when a significant change is observed for the measured RTT with a neighbor, the $loss$ factor can be reset to zero in order to let the node move again. But, what is a "significant change"? It is difficult to define a threshold.

3.6 Conclusion

In this chapter, we have described existing RTT estimation mechanisms and we have chosen one of them (Vivaldi) to develop our estimation-based overlay routing service. Vivaldi has interesting properties: it is fully distributed, it consumes only a small amount of resources to compute its estimations and it has been successfully tested at the scale of the Internet. Moreover, a network simulator called P2PSim provides an implementation of Vivaldi and is publicly available. We will use that simulator for our experiments. We have also discussed Vivaldi's parameters (neighbors, type of embedding space and algorithm's parameters) and we have chosen one value that seems suitable for each of them.

Using an estimation mechanism like Vivaldi should allow us to design a scalable overlay routing mechanism. Indeed, with an estimation-based approach, the measurement cost is only $O(n \times m)$ (compared to $O(n^2)$ with RON) where n is the number of nodes and $m \lll n$ is the number of neighbors used by each node. The communication cost is a little bit more difficult to estimate. For our experiments, we simply consider that each node sends its coordinate to all the other nodes (like each node sends its measurements results to all the other nodes in RON). With such simple communication strategy, we have a number of messages $O(n^2)$ and a size of message $O(d)$ where d is the number of dimensions of the embedding space. So, the total communication cost is $O(n^2 \times d)$. Since $d \lll n$, this is better than RON's communication cost (which is $O(n^3)$), but it is still $O(n^2)$ and it is not really scalable. However, it should be possible to reduce it using strategies like the one presented in [77] for RON. We have not considered in detail the problem of data dissemination during our study and this is still a future work.

So, we have an RTT estimation mechanism and we intend to use it to find good alternative routes in the Internet. A basic solution, could be to replace the measurements by estimations in RON's approach of the problem. In other words, if we want to find an one-hop routing shortcut for a path AB, then, instead of checking if

$$RTT(A, C) + RTT(C, B) < RTT(A, B)$$

for each potential relay node C, a simple solution could be to check if

$$EST(A, C) + EST(C, B) < EST(A, B)$$

where $EST(X, Y)$ is the estimated RTT between the nodes X and Y. If that would work, we would have a way to find routing shortcuts with a small measurement cost compared to RON's approach (we need only the few measurements done by Vivaldi to compute the estimations). However, this is not so simple and we will see in next chapter that it is impossible to find routing shortcuts in this way.

Chapter 4

Triangle Inequality Violations and ICS

Abstract

In this chapter, we will study the well-known problem of triangle inequality violations (TIVs) encountered by Vivaldi and the other classical ICSs. Each ICS that tries to embed RTTs in a metric space make the assumption that the triangle inequality rule holds for RTTs in the Internet. Unfortunately, we will see that this is a wrong assumption and that TIVs have a significant impact on the behavior of Vivaldi and on the accuracy of the estimations produced by Vivaldi. Following these observations we developed some criteria to detect TIVs by analysing the behavior of Vivaldi and we will discuss these criteria at the end of the chapter.

4.1 Triangle Inequality Violations (TIV)

4.1.1 Definition

In the previous chapter, we have seen that Vivaldi (and most Internet coordinate systems) embeds RTT measurements into a metric space and assigns a coordinate in this space to each node of the network in order to enable accurate and cheap RTT predictions between any pair of nodes in the network. That operation requires a metric space, where, by definition, the triangle inequality holds (see definitions 10 and 11 page 35): for any x, y, z belonging to a metric space we have

$$d(x, z) \leq d(x, y) + d(y, z)$$

where $d()$ denotes the distance in the space. Since the distances between nodes in the metric space represent RTTs between nodes in a network, the triangle inequality must hold for the RTTs in the network. If it is not the case, the RTTs are not embeddable in a metric space. So, in this thesis, we will focus on the triangle inequality violations by the RTTs in networks. Let A, B and C be three nodes. A *Triangle Inequality Violation (TIV)* exists between these nodes when the triangle inequality does not hold for the RTTs between these nodes, *i.e.* when

$$
\begin{aligned}
& RTT(A,B) > RTT(A,C) + RTT(C,B) \\
\text{or}\quad & RTT(A,C) > RTT(A,B) + RTT(C,B) \\
\text{or}\quad & RTT(C,B) > RTT(A,B) + RTT(A,C)
\end{aligned}
$$

In order to simplify this definition, we introduce the concept of *triangle*. A triangle ACB is a set of three nodes A, B and C, where, by convention, AB is the longest edge of the triangle. With that convention, it is enough to consider the inequality with respect to the longest edge AB of the triangle and the definition of a TIV becomes

Definition 12. *Let ACB be a triangle. If*

$$RTT(A,B) > RTT(A,C) + RTT(C,B)$$

then ACB is called a Triangle Inequality Violation (TIV).

Definition 13. *A path AB is a* TIV base *if and only if there exists at least one node C such that ACB is a TIV.*

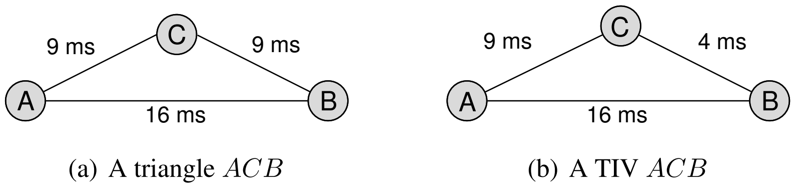


Figure 4.1: **Examples of triangle and TIV.** Figure 4.1(a) shows a triangle ACB: AB is the longest edge and the triangle inequality holds ($16\,ms < 9\,ms + 9\,ms$). Figure 4.1(b) shows a TIV ACB: AB is the longest edge and the triangle inequality is violated ($16\,ms > 9\,ms + 4\,ms$).

4.1.2 TIVs and estimation errors

Since an ICS maps RTTs measured in a network to distances in a metric space, an exact mapping of the RTTs requires that the triangle inequality holds for the RTTs. If it is not the case, it is impossible to map the RTTs to distances in the metric space without estimation errors. The problem is illustrated by example 2 for 2-dimensional Euclidean space, but this is true for any metric space.

Example 2. *First consider the topology presented in figure 4.2. The topology is composed of three nodes A, B and C and the RTTs between these nodes do not violate the triangle inequality. Since the RTT between A and B is $16\,ms$, $(0,0)$ and $(16,0)$ are possible adequate coordinates (among others) for A and B because the distance in the space between these coordinates is equal to 16. For node C, we have $RTT(A,C) = 9\,ms$ and $RTT(C,B) = 9\,ms$. So, an adequate coordinate for node C must be at a distance 9*

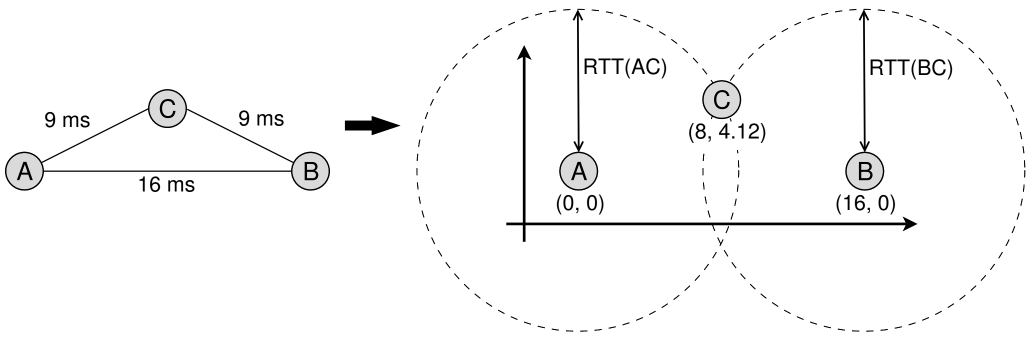


Figure 4.2: **Example of embedding without TIV.** If the triangle inequality holds for RTTs, it is possible to find one coordinate for each node such that the distance between the nodes correspond exactly to the RTTs.

of A's coordinate (i.e. somewhere on the circle drawn around A's coordinate) and at a distance 9 of B's coordinate (i.e. somewhere on the circle drawn around B's coordinate). Since there are two points of intersection between these circles, there are two adequate coordinates, and we can choose $(8, 4.12)$. As explained in section 3.3.4, to know which of these two coordinates is adequate with respect to the coordinates of all the nodes of the topology (not represented in figure 4.2), we need at least one more measurement. However, this is not the problem in this example: we just want to show that, in general, when the triangle inequality holds for RTTs, it is possible to find coordinates providing RTT estimations that correspond exactly to the measured RTTs.

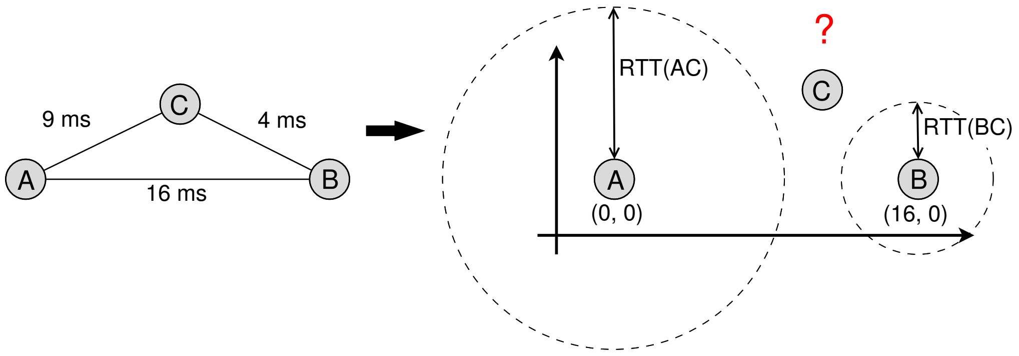


Figure 4.3: **Example of embedding with a TIV.** If the triangle inequality does not hold for RTTs, it is impossible to find coordinates for the nodes that do not generate estimation errors.

Now, consider figure 4.3. The topology is still composed of three nodes A, B and C but the triangle inequality is violated by the RTTs measured between these nodes. Indeed, we have $RTT(A,B) > RTT(A,C) + RTT(C,B)$. Like in figure 4.2, we try to find a good coordinate for each node. The coordinates $(0,0)$ and $(16,0)$ are still a good choice for the nodes A and B. The problem appears when we try to find a coordinate for node C. We have $RTT(A,C) = 9\,ms$ and $RTT(C,B) = 4\,ms$. So, an adequate coordinate for node C must be at a distance 9 of A's coordinate (i.e. somewhere on the circle drawn around A's coordinate) and at a distance 4 of B's coordinate (i.e. somewhere on the circle drawn around B's coordinate). The problem, is that these two circles are disjoint:

if we put a distance of 16 *between* A*'s coordinate and* B*'s coordinate, it is impossible to find a coordinate for* C *which is, in the same time at a distance* 9 *of* A*'s coordinate and at a distance* 4 *of* B*'s coordinate. In other words, if the triangle inequality does not hold for the RTTs, estimation errors are unavoidable.*

Example 2 shows that violations of the triangle inequality automatically leads to estimation errors. So triangle inequality violations are a real problem for Vivaldi and lots of other Internet coordinate systems. The impact of TIVs on the quality of the estimations will be discussed in section 4.4.

4.1.3 Other sources of estimation errors

It is well known that TIVs is a source of embedding errors for ICSs [37, 47] but this is not the only one. In this section, we will investigate three possible sources of estimation errors for Vivaldi which can appear even in topologies where the triangle inequality holds.

Node blocked in a local minimum

Remember that Vivaldi adjusts the coordinate of each node to minimise the estimation errors for the paths between the node and its neighbors. At each round, a node chooses randomly one of its neighbors, performs a RTT measurement with that neighbor and calls algorithm 2 to update its coordinate: depending of the current estimation error (computed thanks to the measurement result and the coordinates), the node will make a small step towards its neighbor or away from its neighbor in order to reduce the estimation error. Remember also that the initial state of Vivaldi is the chaos: all the nodes are located at the origin of the metric space and move in random directions before finding an accurate position in space with respect to the coordinates of their neighbors. So, Vivaldi includes random choices and particular sequences of random choices can lead to situations where a node is blocked in position corresponding to a local minimum for the estimation error. A simple example of such problem is given in example 3.

Example 3. *In the first part of example 2, we have seen in figure 4.2 that there are two good positions in the space for* C *with respect to* A*'s and* B*'s positions and that (at least) a third neighbor is required to decide which coordinate is the best coordinate for* C *between* $(8, 4.12)$ *and* $(8, -4.12)$*. As shown in figure 4.4, we add a third neighbor* D *that indicates that the best coordinate for* C *is* $(8, -4.12)$*. Even though the best position for* C *is* $(8, -4.12)$*, particular sequences of random choices in Vivaldi can push* C *towards* $(8, 4.12)$*. For example, suppose that the initial random moves push* C *above the line formed by* A *and* B*. Then, if the random selection mechanism of a neighbor at each round uses essentially* A *and* B *to compute* C*'s coordinate, there is a probability that* C*'s coordinate converges towards* $(8, 4.12)$*. Indeed, once* C*'s coordinate is approximately* $(8, 4.12)$ *and once* C *has done a few measurements with* A *and* B*, its local error will be low and the algorithm's adaptive timestep becomes a small value. Consequently, each time* C *does a measurement with* D*, it will make a small step towards* $(8, -4.12)$ *but it is not sufficient to push* C *on the other side of the line. So, each next measurement with*

A or B will push C immediately back towards $(8, 4.12)$. In such case, C is blocked with coordinates that minimizes the estimation errors with respect to A's and B's positions but not with respect to D's position. In other words, it is a local minimum but not a global minimum.

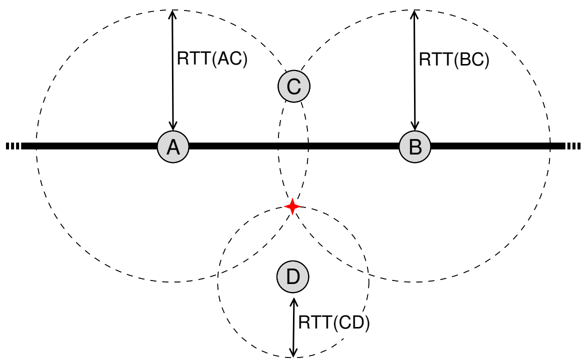


Figure 4.4: **Example of node blocked in a local minimum.** The position of node C is correct with respect to the positions of its neighbors A and B but not with respect to the positions of its neighbor D and of all the other nodes in the topology.

The scenario presented in example 3 has only a small probability to happen[1] but it can happen. A solution to reduce the probability of such a scenario is to take more than the minimum number of neighbors required for a d-dimensional space ($d + 1$ neighbors). Indeed, in example 3, if C uses, for example, 10 neighbors instead of 3 neighbors, the probability of choosing only A and B for the first updates of C's coordinate is reduced: since there are more neighbors, there are more chances to choose a neighbor other than A or B that would indicate that C is located on the wrong side of the line formed by A and B.

Bad selection of neighbors

A bad initial selection of neighbors (when the node selects the set of nodes it will use as neighbors), can also lead to bad coordinates. Indeed, in section 3.3.4, we stated that at least $d + 1$ neighbors are required to define a unique position for each node in a d-dimensional space. As shown in figure 4.5, we should have mentioned that these neighbors must have linearly independent coordinates: in figure 4.5, even if C uses three neighbors (A, B and D) in a 2-dimensional space, it has not enough information to discover its optimal coordinate. Using more than the minimum number of neighbors reduces also the probability of such problems.

[1]It requires a bad situation at the end of the initial chaos and a bad sequence of neighbors used to update the coordinate.

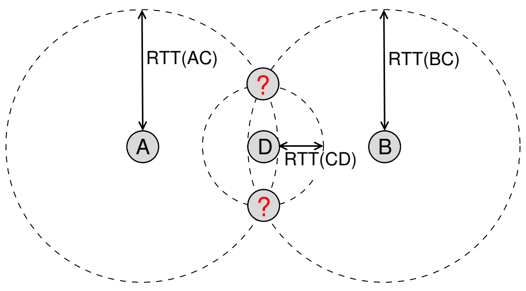


Figure 4.5: **Example of bad neighbor selection.** If C uses A, B and D as neighbors, C has not enough information to find its best coordinate because its neighbors have linearly dependent coordinates.

Non embeddable TIV-free topologies

We have seen that it is impossible to embed a topology in a metric space if the triangle inequality does not hold for the RTTs in that topology. There exist also non-embeddable topologies where the triangle inequality holds. A simple example is given in example 4.

Example 4. *A simple example of a TIV-free topology that is impossible to embed in a given space is a topology where all the RTTs between the nodes are identical. It is clear that a topology where all the RTTs are equal to X contains no TIV. However, depending of the number of nodes it contains, such topology is not embeddable in every metric space. Figure 4.6 shows the problem for a 2-dimensional space. Figure 4.6(a) gives one possible embedding if the topology contains three nodes. This figure shows also that adding a fourth node is impossible because there is no intersection between the three circles of diameter X. In particular, figure 4.6(b) shows that building a square is not an accurate solution (it produces estimation errors for the paths AD and CB). The only solution to add a fourth node at a distance X of the three others is to add a third dimension to space and to build a tetrahedron. But, in a 3-dimensional space, it will be impossible to add a fifth node at a distance X of the others.*

If we want to generalize the problem presented in example 4, we can say that an n-dimensional space is required to embed a topology of $n+1$ nodes where all the RTTs are identical. We have discussed the problem of identical RTTs but it is easy to understand that Vivaldi will also have difficulties to embed topologies where all the RTTs are similar. We will see in chapter 6 that being aware of such potential embedding problems is quite important.

4.2 TIVs and Routing shortcuts

By definition, a one-hop routing shortcut is a TIV: a triangle ACB is a TIV if and only if C is a shortcut for the path AB. Consequently, the simple solution proposed in the conclusion of the previous chapter to find routing shortcuts using estimations does not work. Indeed, even if C is the shortcut for a path AB, the inequality

$$EST(A,C) + EST(C,B) < EST(A,B)$$

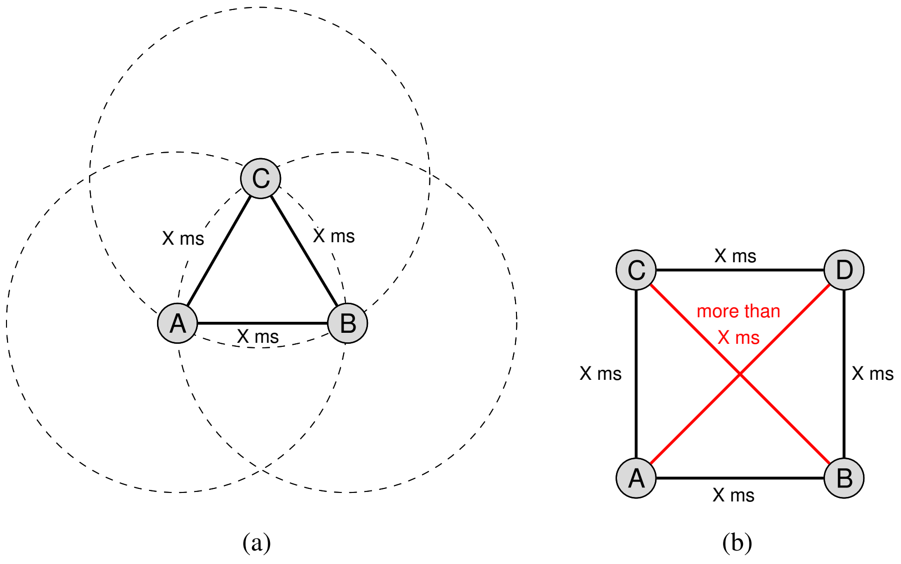


Figure 4.6: **Example non-embeddable TIV-free topology.** In this example, we try to embed a topology where all the RTTs are equal to X in a 2-dimensional space. Figure 4.6(a) shows that it is impossible to embed more than three nodes because there is no suitable position for a fourth node. In particular, figure 4.6(b) shows that building a square does not provide perfect estimations.

will never hold since estimated RTTs are distances in a metric space for which the triangle inequality always holds. In other words, discovering routing shortcuts using estimations only is impossible in a metric space. We will investigate routing shortcuts detection methods combining estimations and measurements in chapter 5. In the current chapter, we will focus only on estimations. Indeed, we know that routing shortcuts (*i.e.* TIVs) will generate unavoidable estimation errors. So, it could be interesting to know if these estimation errors have specific properties. If they have such properties, it could be possible to detect routing shortcuts by analysing the estimation errors. That is what Lumezanu *et al.* proposed to do in [48]. We performed a similar study at the beginning of our work and our results are presented in section 4.4.

4.3 TIVs in the Internet

Before doing anything else, it is important to know if TIVs are common in the Internet or if they are rare and negligible. It is important for ICSs (to know if the embeding problem caused by TIVs is negligible or not) but, it is especially important for our overlay routing research: if there are only a few TIVs in the Internet, that means that overlay routing is useless because there are only a few routing shortcuts in the Internet. Since previous studies (*e.g.* [73]) have demonstrated that overlay routing can significantly improve routing performance, we already know that the number of TIVs in the Internet is not negligible. In section 4.3.1, we analyse why there are TIVs in the Internet. Then, in section 4.3.3 we study the distributions of TIVs existing in the Internet, and we characterize their severity using different metrics.

4.3.1 Sources of TIVs in the Internet

In this section, we try to explain why there are TIVs in the Internet. A few years ago, Zheng *et al.* [85] have shown that TIVs are not measurement artifacts but a natural consequence of Internet routing policies. Through examples based on real measurements, they explain how Internet routing policies generate TIVs.

In section 2.1.1, we have seen that intra-domain routing consists in finding shortest paths with respect to configured link weights. If these weights do not correspond to the RTTs, the path with the smallest RTT is not necessarily selected as the default Internet path and TIVs are created. However, since ISPs try to optimize the routing inside their own networks, weights and RTTs are generally somewhat correlated. Thus, TIVs caused by intra-domain routing policies are generally negligible compared to the TIVs caused by inter-domain routing policies.

BGP routing policies have been discussed in section 2.1.2. Many things in the BGP behavior may lead to situations where the triangle inequality is violated. Zengh *et al.* [85] showed that, in specific situations, TIVs can be caused by hot potato routing, by interactions between inter-domain and intra-domain routing, by the fact that BGP considers only AS paths and does not make any difference between large ASes and small ASes, *etc*. But an important source of TIVs is the commercial relationships between ASes. Example 5 illustrates that problem.

Example 5. *To illustrate how commercial relationships between ASes can lead to TIVs, we take the example of network used in chapter 2. That network is represented in figure 4.7. In this figure, the symbol "\$" denotes a customer-provider relationship between*

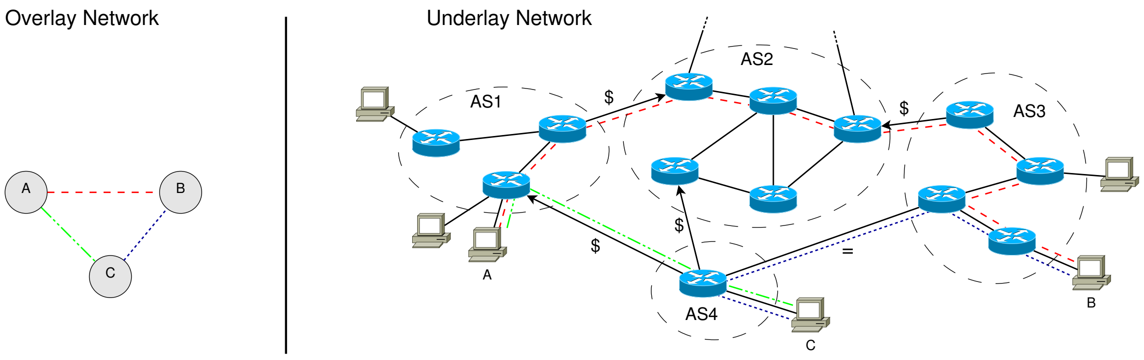


Figure 4.7: **Example of TIV caused by economic relations between ASes.** If we simply consider that each Internet link has an RTT of $1\,ms$, then ACB is a TIV.

two ASes (the arrow indicates who is the provider) and the symbol "=" denotes a peering relationship between two ASes. The routes chosen by BGP depend of the commercial relationships:

- *Since AS_4 is a customer of AS_1, it is easy to see that the route between node A and node C uses the corresponding customer-provider link (route composed of 3 hops).*
- *In the example, AS_4 and AS_3 are both customers of AS_2 but a peering link is established between these two ASes. By definition, that link can only be used for*

traffic generated by a source located in AS_3 (resp. AS_4) or one of its customers (if any) and destined for a node located in AS_4 (resp. AS_3) or one of its customers (if any). So, the path between node C and node B uses the peering link (route composed of 4 hops).

- *The path between node A and node B cannot go through AS_4. Indeed, AS_4 is a customer of AS_1. By definition, a provider can send traffic to one of its customers only if the traffic is destined for this customer (or its customers thereof, if any). Since AS_3 is not a customer of AS_4, AS_1 cannot send to AS_4 the traffic destined for AS_3. Since AS_1 and AS_3 are both customers of AS_2, the route between node A and B will go through AS_2 (route composed of 10 hops).*

For the example, simply consider that making 1 hop takes $1\,ms$. Then ACB is a TIV because

$$\left.\begin{array}{l} RTT(A,B) = 20\ ms \\ RTT(A,C) = 6\ ms \\ RTT(C,B) = 8\ ms \end{array}\right\} \Rightarrow RTT(A,B) > RTT(A,C) + RTT(C,B)$$

Example 5 shows that even in simple topologies with four ASes, commercial relationships between ASes can cause TIVs. It is a small TIV and the corresponding routing shortcut is not really interesting: the alternative path between A and B is only $6\,ms$ shorter than the default path. But it is a small example where AS_1 and AS_3 are customers of the same provider. So, it was not necessary to go high in the ASes hierarchy to find a route between AS_1 and AS_3. If we consider topologies at the scale of the Internet, there are some cases where it is necessary to go very high in the hierarchy to find a route between two points. In such cases, finding routing shortcuts can be a lot more interesting.

In situations like the one presented in example 5, C is a routing shortcut for the path AB but using C as relay to go from A to B is a violation of the routing policies. Indeed, that means that traffic between A and B is routed through AS_4 and that is forbidden by the peering relationship established between AS_4 and AS_3. So, overlay routing may lead to situations where the alternative routes violate the rules established by the ISPs. Consequently, it is clear that ISPs have only a low interest in participating to such overlay (the only one being the improvement of their customers' satisfaction). However, a recent study [49] performed by Lumezanu *et al.* has shown that, even if TIVs are caused by suboptimal route choices, a significant part of the alternative routes corresponding to shortcuts do not violate the routing policies. These routes are simply considered as less efficient by the routing protocols and are not chosen as the default Internet route. Consequently, both end-users and ISPs could take advantage of paths improvements: in many cases, it is thus possible to reduce the delays experienced by the users without violating the routing policies.

4.3.2 Severity metrics

Previous studies [74, 85, 37] have reported characteristics of TIVs in the Internet delay space by triangulation ratio distribution and the fraction of triangles that suffer from TIVs.

We do something similar and we propose two basic characterizations of the *severity* of TIVs. The first one is the *relative gain* and is defined as follows:

Definition 14. *Let ACB be a TIV. The* relative gain *provided by the TIV is*

$$G_r = \frac{RTT(A,B) - (RTT(A,C) + RTT(C,B))}{RTT(A,B)} \tag{4.1}$$

G_r ranges from 0 (minimum severity) to 1 (maximum severity). The relative gain is an interesting metric, but it may be argued that, for small triangles, a high relative gain may not be so critical. For example, for a path AB with $RTT(A,B) = 20\,ms$, $G_r = 10\%$ represents only an improvement of $2\,ms$. Therefore we also define a second metric called the *absolute gain*, which is defined as follows:

Definition 15. *Let ACB be a TIV. The* absolute gain *provided by the TIV is*

$$G_a = RTT(A,B) - (RTT(A,C) + RTT(C,B)) \tag{4.2}$$

G_a ranges from $0\,ms$ (minimum severity) to the diameter of the network (*i.e.* the maximal RTT between any two points in the network).

In the sequel, we will refer to specific severity thresholds and select TIVs whose severities are above them. We can select all TIVs such that $G_r \geq th_r$, or $G_{ar} \geq th_a$, or even when both thresholds are exceeded. This way, we can ignore TIVs which do not provide an important gain: if the difference between the direct path and the alternative path is small, TIVs do not interest us for two reasons. Firstly, from an ICS point of view, they are "almost" embeddable and they do not generate large estimation errors. Secondly, from overlay routing point of view, they are not interesting shortcuts.

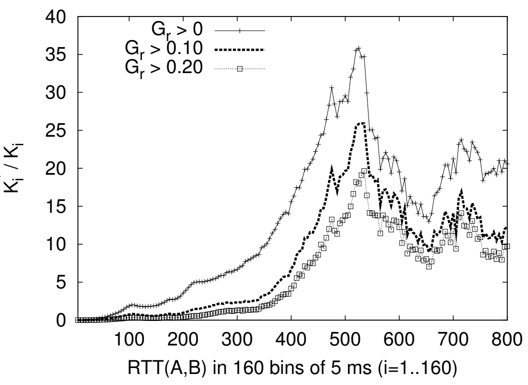
Since our goal is to eliminate all the negligible TIVs, considering simultaneously both thresholds is important. Indeed, we know that it is difficult to define a threshold on the relative gain to decide if a TIV is negligible or not because a high relative gain may not be critical for small triangles. That is why we introduced the absolute gain. But, with the absolute gain, it is also difficult to define a threshold: a threshold suitable for small triangles will not be suitable for large triangles. For example, a TIV providing an absolute gain of $20\,ms$ is not negligible for a path of $100\,ms$, but it is negligible for a path of $1\,s$. Considering simultaneously both thresholds solves the problem: small triangles are constrained by the threshold on the absolute gain and large triangles are constrained by the threshold on the relative gain.

4.3.3 Analysis of triangle inequalities in the Internet

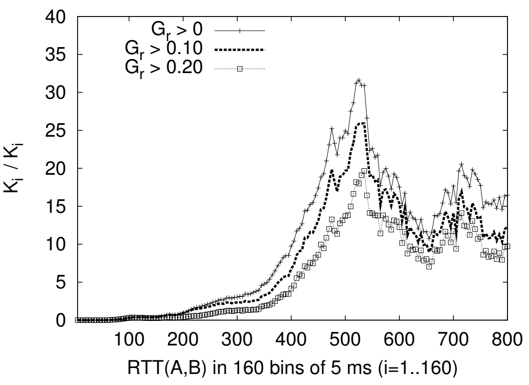
In this section, we study through different metrics the violations of the triangle inequality in the Internet, and we characterize their severity and distribution according to path lengths. To model Internet latencies, we used the P2PSim and the Meridian data sets described in section 3.4. We first define some notations. Let K be the total number of triangles in the two data sets. For the P2PSim data set, we found $K = 854,773,676$, of which $105,329,511$ (representing 12%) are TIVs. For the Meridian data set, we found

$K = 2,598,842,308$ where 23.5% of these triangles are TIVs. We divide the whole range of RTTs in the P2PSim (resp. Meridian) data set into 160 (resp. 600) equal bins of $5ms$ each. Let K_i be the number of triangles in the ith bin. By convention, we say that triangle ACB is in bin i if its longest edge AB is in that bin. Let K'_i be the number of TIVs in the ith bin.

We begin our analysis by showing the proportion of TIVs is each bin, namely K'_i/K_i, for different severity thresholds. Figure 4.8(a) (resp. figure 4.9(a)) only considers relative TIV severities (as $th_a = 0\,ms$) for the P2PSim (resp. Meridian) data set. In figure 4.8(b) (resp. figure 4.9(b)), we filter out TIVs whose absolute severity is below $th_a = 20\,ms$ (resp. $th_a = 15\,ms$) for the P2PSim (resp. Meridian) data set. All these curves have basically the same shapes. We can see clearly that large triangles (say above $400\ ms$) are more likely (severe) TIVs.

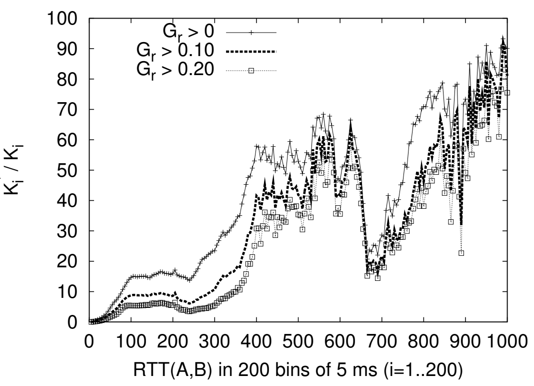


(a) $G_a > 0\,ms$

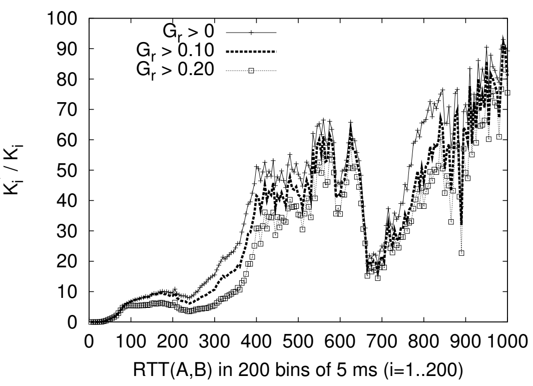


(b) $G_a > 20\,ms$

Figure 4.8: Proportion of TIVs in each bin for various TIV severity levels for P2PSim data set.



(a) $G_a > 0\,ms$



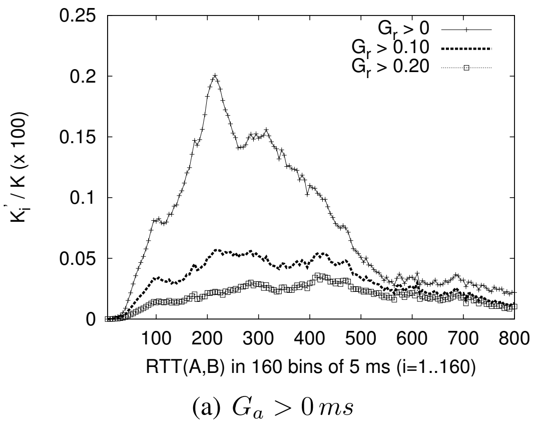
(b) $G_a > 15\,ms$

Figure 4.9: Proportion of TIVs in each bin for various TIV severity levels for Meridian data set.

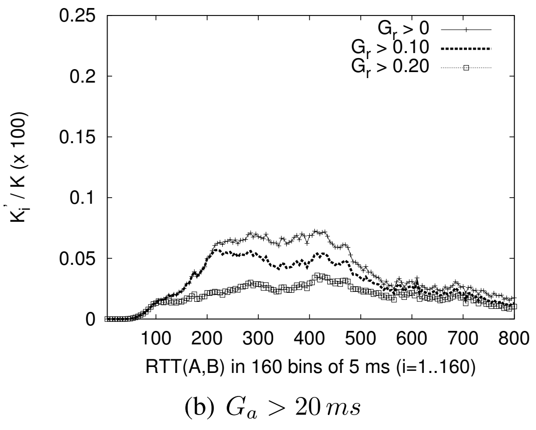
Let P_i the probability for a triangle ACB, chosen at random in the data set, to be (a) in the bin i and (b) to be a (severe) TIV, namely:

$$P_i = \frac{K_i'}{K_i} * \frac{K_i}{K} = \frac{K_i'}{K} \tag{4.3}$$

Obviously, P_i is simply the number of TIVs in the bin i divided by the total number of triangles. The distribution of TIVs in the P2PSim (resp. Meridian) data set is depicted in figure 4.10 (resp. figure 4.11). As expected, figure 4.10 is less conclusive than figure 4.8, but it still shows that few severe TIVs are found in small triangles, that is below $100\,ms$. A similar behavior can be observed in figure 4.11 where the edges shorter than $60\,ms$ cause slight violations. Moreover, the TIV severity of edges has an irregular relationship with their lengths. For instance, in figure 4.11 the TIV severity has a peak for the edges around $80 - 100\,ms$.



(a) $G_a > 0\,ms$



(b) $G_a > 20\,ms$

Figure 4.10: Distribution of TIVs in P2PSim data set for various severity levels.

These observations have motivated our hierarchical approach of Vivaldi (see chapter 8). If we create clusters whose diameters do not exceed too much $100\,ms$, we may expect much fewer severe TIVs in each cluster, which is likely to improve the accuracy of intra-cluster coordinate systems. However, so far, we only know that TIVs are not negligible in the Internet. Their impact on the coordinate embedding remains hypothetical. We will quantify this impact on Vivaldi in section 4.4.

4.3.4 Triangle inequality variations in the Internet

As major part of the studies on TIVs, our study has been made on aggregated latency data sets that combine measurements taken at different times over long periods. Such data sets do not capture the variations of triangle inequalities: depending on the way they are built, such data sets may contain TIVs which were temporary in the network or, reversely, they may miss some long-lived TIVs. That problem has been recently studied by Lumezanu *et al.* [50]. That study essentially shows that TIVs are not illusions of measurements (it

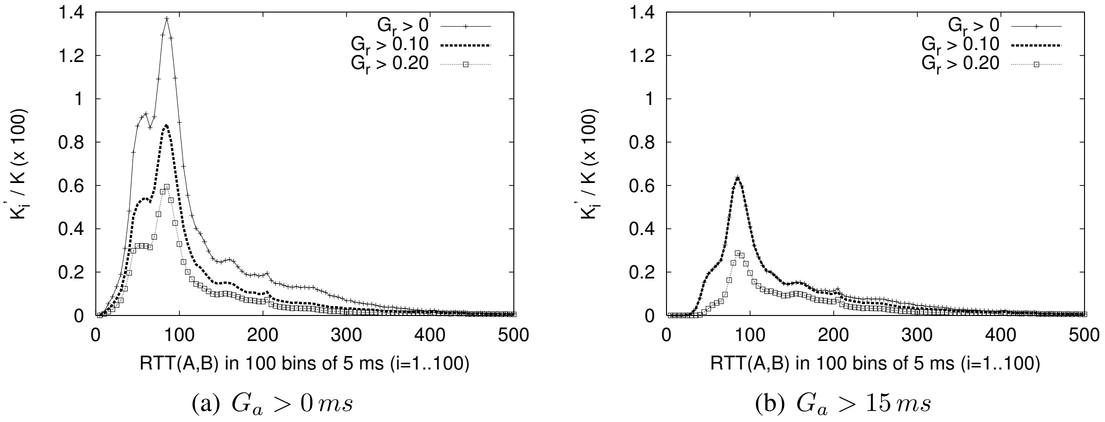


Figure 4.11: Distribution of TIVs in Meridian data set for various severity levels.

confirms the results presented in [85] and [49]) and it shows that TIVs vary with time in the Internet.

They compared the TIVs obtained in their measurement results over a long period of time to the TIVs existing in an aggregated latency matrix computed using the median of the measurements. They observed that, at no point during the measurement process, the number of TIVs has been lower than the number of TIVs in the aggregated data set. That means that median values reveal fewer TIVs than there are in the network. They also observed that scenarios where median values create a TIV that does not exist in the measurements is extremely infrequent. These observations show that our data sets do not represent exactly the Internet: in reality, there are more TIVs than expected by analysing the P2PSim data set[2]. Since our data sets do not model exactly the Internet, these observations may invalidate the results presented in section 4.3.3. However, there is no problem with the other results that will be presented in this thesis: even if these data sets do not represent exactly the Internet, they still represent large networks containing a large number of TIVs. So, they represent a "real" situation. We have just to keep in mind that the Internet may contain even more TIVs than that.

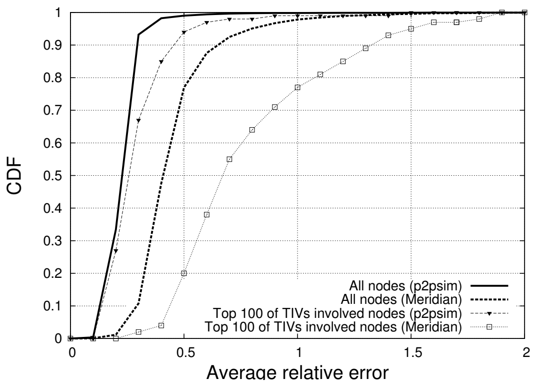
Another interesting result presented in [50] is that a significant part of the TIVs are long-lived and 18% of them have even a longevity of more than 5 hours. That observation is interesting for overlay routing because it implies that it will not be necessary to change shortcuts frequently to improve the routing efficiency. Note that this observation is logical: previous studies have shown that TIVs are mainly caused by bad inter-domain route choices and, since these routes are quite permanent, the TIVs are also quite permanent. However, some short-lived TIVs may still be caused by temporary congestions or other problems.

[2]The major part of the results presented by Lumezanu *et al.* are only valid for the P2PSim data set. The P2PSim data set has been computed using the median values of RTTs observed over a long period of time. The Meridian data set does not. The Meridian data set has been computed using the minimum of the RTTs observed over a long period of time. The article briefly investigates other methods than the median to compute aggregated data sets but the results are less detailed.

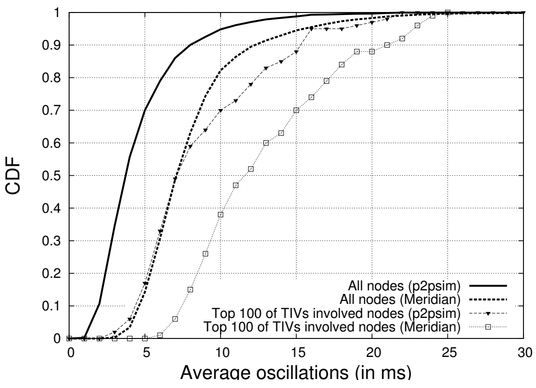
4.4 Impact of TIVs on Vivaldi

In this section, we present the results of an extensive simulation study of the Vivaldi system. For the simulation scenarios, we used the P2PSim discrete-event simulator [64], which comes with an implementation of the Vivaldi system. The choices of parameters for Vivaldi have been discussed in chapter 3. The only difference with the choices presented in chapter 3 is that we used a 2-dimensional Euclidean space instead of the 10-dimensional Euclidean space used for the other results presented in this thesis. There is no particular motivation to use a low-dimensional space for these experiments. The explanation is simply that these results have been obtained at the beginning of our research, when we had still not made the choice to work with a 10-dimensional space[3].

In order to observe the impact of TIVs on Vivaldi nodes, and particularly on the embedding performance, we could define the notion of TIVs involvement through different considerations. We consider a node to be more or less involved in TIVs by counting the number of times it belongs to a TIV. The more node C appears in TIVs A_iCB_j, the more C is considered involved into TIVs situations.



(a) CDF of Average relative errors



(b) CDF of Oscillations

Figure 4.12: Impact of TIV severity on the embedding for P2PSim and Meridian data set.

In figure 4.12(a), considering the P2PSim and Meridian data sets, we plot the Cumulative Distribution Function (CDF) of the average relative error (ARE) of the top 100 nodes involved in TIVs. We compare such distribution to the CDF of all nodes' relative errors in a Vivaldi system. Note that the ARE is computed for each node as the average prediction errors (with respect to *all* the nodes) yielded by the Vivaldi system at the last

[3]Even though we have not performed the preliminary study presented here in a high dimensional space, we can ensure that the impact of TIVs on Vivaldi is roughly the same in a 10-dimensional space as in a 2-dimensional space. Indeed, in section 4.6, we will propose TIV detection mechanism based on the observations reported here (TIVs cause estimation errors, coordinate oscillations, *etc.*), and these detection mechanisms have been tested successfully using a 10-dimensional space.

tick of our simulations.

$$ARE(A) = \frac{\sum_{(B \neq A) \in S} \frac{|RTT(A,B) - EST(A,B)|}{RTT(A,B)}}{|S| - 1}$$

where S is the set of all nodes in the system. Figure 4.12(a) shows that, while for the distribution of errors on all the nodes of the system, more than 90% of P2PSim nodes (resp. 70% Meridian nodes) have an ARE less than 0.3 (resp. 0.5), this percentage falls down to only 50% (resp. 20%) when considering the most involved nodes in TIVs. The coordinate computations at the level of these involved nodes is spoiled out.

It is also worth observing the variation of coordinates in the Vivaldi system. In fact, even though the system converges in the sense that the relative errors at each node stabilizes, these errors could be so high that a great variation of the coordinate of a node barely affects the associated error. We can define such coordinates oscillation as the distance between any two consecutive coordinates. The average oscillation values are computed as the average of the oscillations during the last 500 ticks of our Vivaldi simulation. Figure 4.12(b) shows the CDF of these average oscillations, comparing again the distributions through all nodes and of the top 100 nodes involved in TIVs. We clearly see that the impact of TIVs can be considered as very serious with nodes involved in more TIVs seeing a large increase in their average oscillations values.

In light of these observations on the serious impact of TIVs on the coordinates embedding, we oriented our research on two paths. Firstly, we proposed a hierarchical structure of Vivaldi to mitigate the impact of most severe TIVs [7, 29, 6]. In this way, nodes would perform a more accurate embedding at least in restricted spaces (*i.e.* with small distance coverage). The hierarchical approach for Vivaldi will be discussed in chapter 8 of the thesis. Secondly, since TIVs have a serious impact on the embedding, we tried to detect these TIVs by observing the characteristics of the estimations [43, 28]. These results are presented in section 4.6.

4.5 Detecting TIVs

From the overlay routing point of view, it is interesting to be able to detect triangles ACB that are TIVs. That is what Lumezanu *et al.* try to do in the overlay routing mechanism named Peerwise [48]. Their detection criterion is based on the assumption that, since a TIV is not embeddable, there will be estimation errors. This is correct but they also make the assumption that the estimation error is systematically distributed among all the edges of the triangle. As indicated in figure 4.13, they suppose that the TIV base is a little bit under-estimated and that the short edges are a little bit over-estimated in order to embed the nodes in the metric space with a small estimation error on each path. Following that assumption, they consider that ACB is a TIV if $EST(AB)$ is under-estimated and $EST(AC) + EST(CB)$ is over-estimated. We do not agree with the second part of their assumption mainly for two reasons:

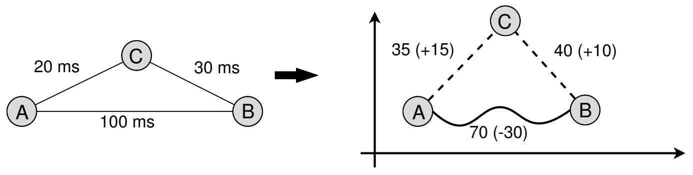
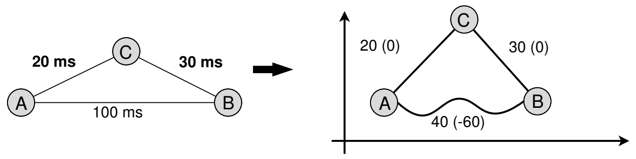


Figure 4.13: **Lumezanu's assumption about estimation errors caused by TIVs.** Numbers in parentheses represent estimation errors. They consider that the embedding error is distributed among all the edges: the short edges are over-estimated (dotted lines) and the base under-estimated (compressed line).

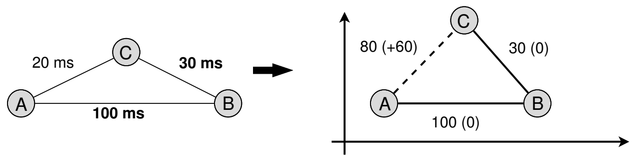
TIVs have effect on the other TIVs' impacts: Our first argument is that the impact of a TIV on the embedding error does not depend only of the TIV itself. It depends also of the other TIVs in the network. For example, add a fourth node D in figure 4.13 and suppose that ADC is also a TIV. Following Lumezanu's assumption, the path AC is over-estimated because it is a short edge in the TIV ACB. But, it is also under-estimated because it is the base of the TIV ADC. Obviously, that is impossible to be simultaneously over-estimated and under-estimated. So, what happens in such situation? It is difficult to say and it leads us to our second argument.

TIV's impact depends of what is measured: Our second argument is that Vivaldi does not try to minimize the estimation errors for all the paths but only for the paths it measures (*i.e.* the paths between the nodes and their neighbors). Thus, the impact of a TIV depends on which edges are measured. Simple examples are given in figure 4.14. For these examples, we suppose that ACB is the only TIV in the topology and that Vivaldi does not monitors all the edges of the triangle (bold values in the figures are measured). Since there are no other TIV than ACB in the topology, it should be possible to find coordinates for A, B and C so that there is no estimation errors for the measured paths. Indeed, if at least one of the edges of the TIV is not measured, its length is infered from the measurements and Vivaldi gives it a "logical" value as estimation: it cannot guess that the path which is not measured is abnormally long or short (*i.e.* that ACB is a TIV). Figure 4.14(a) shows a possible embedding when the two short edges of the TIV are measured but its base is not measured. In such case, nodes find good positions with respect to their neighbors positions and Vivaldi should embed the measured edges without errors. Consequently, only the estimation of the base is impacted by the TIV. Figure 4.14(b) shows a possible embedding when one short edge and the base are measured. In such case, Vivaldi infers a logical value for AC from the measurements and AC is highly over-estimated.

Our simulation results confirm our conjecture about Lumezanu's assumption. In P2PSim and Meridian data sets, more than 70% of the TIV bases are under-estimated but only about 50% of the TIVs' short edges are over-estimated. So, even if the under-estimation seems to be a quite good TIV bases detection criterion (see section 4.6), relying on over-estimations to decide if C is a shortcut for a given path AB will give results similar to a random choice (the probability to take the good decision is only of 50%).



(a) Two short edges measured



(b) Base and one short edge measured

Figure 4.14: **Example of embeddings with partially measured TIVs.** If a TIV is only partially measured, it is impossible to guess that this is a TIV. So, since Vivaldi tries to minimize the errors for the edges that are measured, the embedding error appears mainly on the edges that are not measured.

Moreover, even if their assumption was correct their approach is quite strange. The objective of using a coordinate system to find routing shortcuts is to avoid performing too many measurements. But, they do not only use estimations, they use *estimation errors*: to know if a node C_i is a shortcut for a path AB, they need to check if the path AC_iB is over-estimated. The problem is that computing estimation errors requires measurements. So, to find the best shortcut for a path AB, they need to perform a measurement between A and each C_i and between B and each C_i. If these measurements are available, the coordinate system is useless: it is more reliable to use directly the measurements to decide if a node C_i is a shortcut or not rather than using estimation errors.

Like Lumezanu *et al.*, we tried to detect TIVs by observing Vivaldi's behavior. But, our conclusion is only that the impact of TIVs is very difficult to characterize, even in small topologies where there are only a small number of TIVs. As indicated above, we have noticed that TIVs' impacts are themselves affected by other TIVs and that the impact of one TIV depends of which paths are measured by Vivaldi. All of this makes the detection of routing shortcuts very difficult if we are limited to the estimations provided by an ICS. Even though the detection of routing shortcuts (*i.e.* of TIVs) is difficult, it seems possible to detect TIV bases by observing the behavior of the ICS. For example, we have seen that the under-estimation of a path AB is a quite reliable indication that AB is a TIV base. We will investigate that (and other indicators) in section 4.6.

4.6 Detecting TIV bases

Detecting TIV bases means being able to say if a given path AB is a TIV base or not. In other words, for a given path AB, we intend to be able to say if there exists at least a node C that is a shortcut for AB. Since, we do not know exactly which node is that node C, the detection of TIV bases is not a answer to our main problem of finding routing shortcuts. However, such mechanism can still be useful:

Overlay routing: If we have a simple way to know if there exists at least one shortcut or not for a given path AB, we can avoid testing each potential C to know if it is a shortcut for AB. In large networks, there can be thousands of potential shortcuts to consider. So, being able to answer directly that there is no shortcut when this is the case is interesting.

Excluding TIVs from the neighbor relationship: Wang *et al.* [81] propose to improve the accuracy of the embedding by avoiding to select a neighbor for one node if the path between these two nodes is a TIV base. Obviously, such work also requires a reliable TIV base detection mechanism.

4.6.1 Methodology

We intend to compare multiple TIV base detection criteria. These criteria are based on the characteristics (estimation error, *etc.*) of the estimations provided by Vivaldi. For the simulation scenarios, we used the P2PSim discrete-event simulator [64]. During our simulations of Vivaldi, each node computes its coordinate in a 10-dimensional Euclidean space using 32 neighbors. For each path, the characteristics used by the TIV base detection criteria are computed based on the coordinates provided at the end of the Vivaldi simulations. Using these characteristics we performed detection tests with multiple detection thresholds (when it is relevant) for each detection criterion. To compare the detection results, we need a way to characterize the performance of a detection test and a convenient way to compare the performance of different detection tests.

Characterizing the performance of a detection test

To characterize the performance of our detection tests, we use the classical false/true positive/negative indicators. Specifically, a *negative* is a non-TIV base, which should therefore not be reported in the set of suspects that we derive. A *positive* is a TIV base, which should therefore be suspected by the test. The number of negatives (resp. positives) in the population of paths is $\mathcal{P}_N$ (resp. $\mathcal{P}_P$).

A *false negative* is a TIV base that has been wrongly classified by the test as negative, and has therefore been wrongly unsuspected. A *false positive* is a non-TIV base that has been wrongly suspected by the test. *True positives (resp. true negatives)* are positives (resp. negatives) that have been correctly reported by the test and therefore have been rightly suspected (resp. unsuspected). The number of false negatives (resp. false positives, true negatives and true positives) reported by the test is $\mathcal{T}_{FN}$ (resp. $\mathcal{T}_{FP}$, $\mathcal{T}_{TN}$ and $\mathcal{T}_{TP}$).

We use the notion of *false negative rate* (FNR) which is the proportion of all TIV bases that have been wrongly reported as non-TIVs by the test, and $\mathrm{FNR} = \mathcal{T}_{FN}/\mathcal{P}_P$. The *false positive rate* (FPR) is the proportion of all the non-TIV bases that have been wrongly reported as positive by the test, so $\mathrm{FPR} = \mathcal{T}_{FP}/\mathcal{P}_N$. Similarly, the *true positive rate* (TPR) is the proportion of TIV bases that have been rightly reported as TIVs by the test, and we have $\mathrm{TPR} = \mathcal{T}_{TP}/\mathcal{P}_P$. A good detection mechanism must be able to provide a high TPR and, simultaneously, a low FPR.

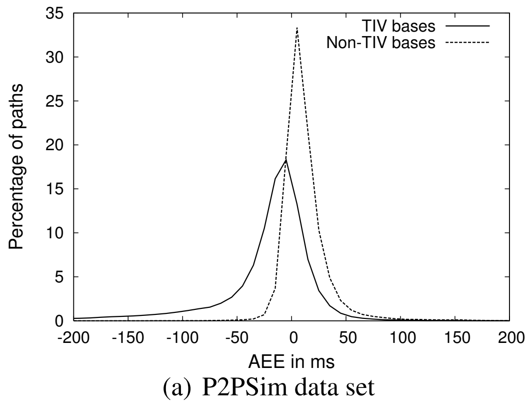
Comparing the performance

Since a good detection mechanism must be able to provide a high TPR and a low FPR, a convenient way to compare detection results is to plot the result of a detection test as a pair (FPR,TPR) in a 2-dimensional Euclidean space. Obviously, the closer to the upper left corner of the graph a point is, the better it is: such points correspond to high true positive rates (i.e. a high proportion of positives being reported as such by the test) for low false positive rates (i.e. a small proportion of negatives incorrectly reported as positives). So, the test generating the point which is the closer to the upper left corner of the graph is the test providing the best result.

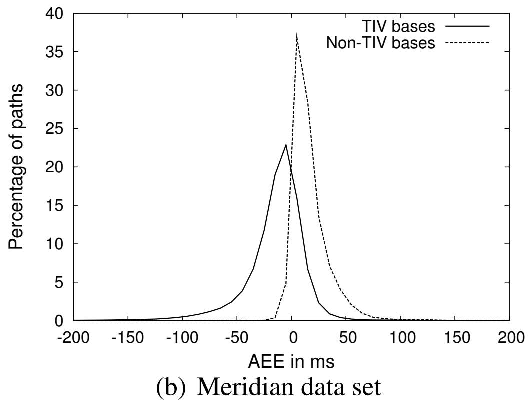
A criterion requiring a detection threshold will generate one point in the graph for each value of the threshold. These points compose a curve called Receiver Operating Characteristic (*ROC*) curve. The point of that curve which is the closest to the upper left corner of the graph is the best value for the threshold.

4.6.2 Detection based on estimation errors

Developing a TIV base detection criterion based on estimation errors is not a new idea. Wang et *al.* [81] showed that there exist a relation between the estimation error and the TIV severity. They observed that if a path AB is a TIV base, it is probably shrunk in the metric space. By comparing the RTTs of our two delay matrices to the estimations obtained with Vivaldi, we found the same trend. Indeed, according to the P2PSim and Meridian data sets, more than 70% of TIV bases are under-estimated (*i.e.* they have a negative estimation error). For each delay matrix, we clusterize paths into two groups: the TIV bases, and the non-TIV bases. Based on the P2PSim (resp. Meridian) data set, figures 4.15(a) and 4.16(a) (resp. figures 4.15(b) and 4.16(b)) show the distributions of paths with respect to their AEE and REE (see definition 8 page 25). We divide the whole range of AEE (resp. REE) into bins equal to $10\,ms$ (resp. 0.05).

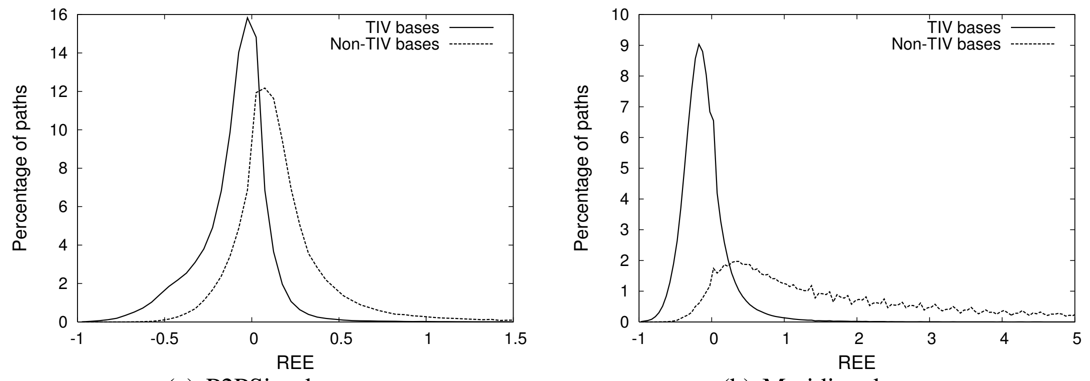


(a) P2PSim data set



(b) Meridian data set

Figure 4.15: Distribution of the paths in function of AEE.



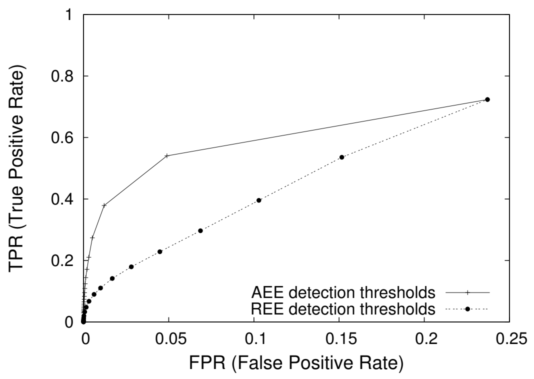
(a) P2PSim data set (b) Meridian data set

Figure 4.16: Distribution of the paths in function of REE.

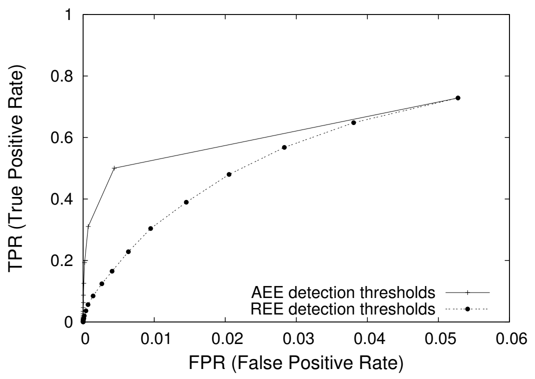
According to figures 4.15 and 4.16, we can see that a criterion based on the estimation errors (as the one proposed in [81]) has a serious drawback: even if the figures confirm that TIV bases are more under-estimated than non-TIV bases, the overlapping of the two curves is important in each case. So, it would be difficult to discriminate the TIV bases.

In order to evaluate the efficiency of a detection based on the under-estimations of the paths, we used multiple detection thresholds based on AEE and REE. For AEE, we consider thresholds varying from $0\ ms$ to $-200\ ms$ by steps of $10\ ms$ and for REE the thresholds vary from 0 to -1 by steps of 0.05. The ROC curves obtained are presented on figures 4.17(a) and 4.17(b) respectively for the P2PSim and the Meridian data sets. Note that these results are obtained by considering all the paths of the delay matrices (this will not be the case in section 4.6.4). Each point on the ROC curves determines the TPR and the FPR obtained with a given detection threshold. For instance, considering the curves obtained with the detection thresholds based on AEE (resp. REE), the first point from the right corresponds to $0\ ms$ (resp. 0), the second corresponds to $-10\ ms$ (resp. 0.05), *etc*. We can see that better results are obtained with a detection threshold based on AEE. Nevertheless, the main drawback of the AEE, is the fact that it depends on the underlying network: if there are only small RTTs the detection threshold should be a small value and, inversely, if there are lots of high RTTs, the detection threshold should be a higher value. According to the relative error metric, we do not have such drawbacks. For this reason, we will only consider REE in the sequel of this study.

These figures confirm also that a criterion based only on under-estimations has a serious limitation: even if we consider all the under-estimated paths (threshold equal to 0 for AEE or REE), we are unable to detect more than 70% of the TIV bases. Moreover, even with a threshold equal to 0, figure 4.17 shows that we still have a low FPR (especially with the Meridian data set). This indicates that we can generalize the detection criterion based on under-estimations and define a detection criterion based on estimation errors (*i.e.* we will consider positive values for the threshold). The results obtained with such criterion will be presented in section 4.6.4 where it is compared to other detection criteria.



(a) P2PSim data set

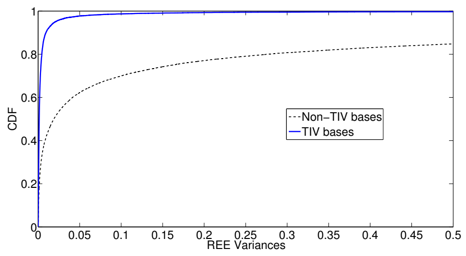


(b) Meridian data set

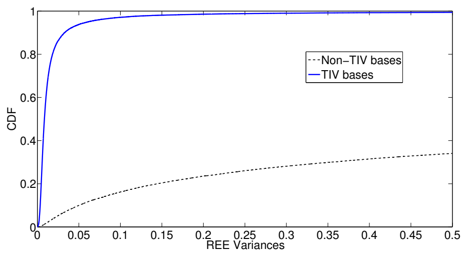
Figure 4.17: ROC curves for the underestimation detection criterion.

4.6.3 Detection based on REE variance

Since a detection criterion based on the basic AEE and REE parameters cannot give satisfactory results, we take into account another parameter. Instead of considering the relative estimation error at a fixed time, a simple alternative is to observe its evolution. For instance, the variance is a metric that can characterize the evolution of the REE with respect to time. To implement a TIV base detection criterion, we compute the REE variances of node pairs during the last 100 ticks of our simulation. Considering the P2PSim (resp. Meridian) data set, figure 4.18(a) (resp. figure 4.18(b)) shows the CDF of the REE variances of the TIV bases and the non-TIV bases. The first observation is that most TIV bases have small REE variances compared to non-TIV bases.



(a) P2PSim data set



(b) Meridian data set

Figure 4.18: CDF of the REE variance.

These findings lead to an easy TIV base detection criterion: if a path has a low REE variance, it is likely to be a TIV base. Since the non-TIV base curve has a gentle slope on figure 4.18(b), such criterion is expected to give very good results (high TPR and low FPR) on the Meridian data set. On the P2psim data set, the results should be less satisfactory. Indeed, we can see on figure 4.18(a) that about 25% of the non-TIV bases have small REE variances. This will probably lead to false positives with a detection criterion based on the REE variance.

In the light of these observations, we propose methods based on REE variance to detect TIV bases. Since these methods require the variance computation, they require the monitoring over a significant period of time of the path for which we want to know if it is a TIV base or not. So, it is very costly to provide a detection mechanism that can be applied to any path in a network. It should be more convenient to restrict the set of paths for which we are able to provide an answer to the paths that are permanently monitored by Vivaldi (*i.e.* the paths between the nodes and their neighbors). Thus, in the sequel of this study, the population of paths concerned by the detection process is limited to the paths monitored by Vivaldi. Since it cannot provide an answer for any given path in the network, such a detection criterion is unsuitable as a first filter for our overlay routing system. However, it can still be useful to improve the selection of the neighbors in Vivaldi like Wang *et al.* [81] propose to do.

So, we consider that each node is maintaining a sliding window (history) of the REE of each path to its Vivaldi neighbors. Each node then computes variances of such REE and ends up, at each embedding step, with a variance vector $\vec{v}$ of m entries (m being the number of neighbors used by the node to compute its coordinate). The basic idea behind our detection methods is to differentiate variances of TIV bases from variances of non-TIV bases. We aim at creating two separate sets of variances and the set that leads to the minimum mean is likely to contain variances of TIV bases.

Detection by ARMA models

In this first approach, our objective is to cluster the variances of the REE using the change point detection method. The key idea is to consider a vector $\vec{v}$ of sorted variances as a time series $v(t)$, to model such series, and then to detect data discontinuities using the model predictions. One of the most suitable models, frequently used in system identification and change point detection is the ARMA (AutoRegressive Moving Average) model [46]. An important advantage of the ARMA model is its ability to analyse the time series by breaking them into homogeneous segments, if there are apparent discontinuities in the time series. The algorithm used for this study follows the approach presented in [39, 46]. In this thesis, we do not want to flood the reader with unnecessary mathematical considerations and we will consider the algorithm as a black box. Just consider that it takes a time series as input and that it indicates the change points it detects in the time series. Interested readers can find more details about how the change points are computed in [28].

Each time the algorithm indicates a change point, we log the instants when such a change occurs. Recall that we consider such time series as series of variances of REE. Hence, when a change occurs at "time" τ, we will consider all $v(t)$, $t < \tau$ as variances of the REE of TIV bases. Obviously, changes may be multiple, but to differentiate variances of TIV bases from variances of non-TIV bases, a node will consider only the first change point detected by the model.

Detection by GMM clustering

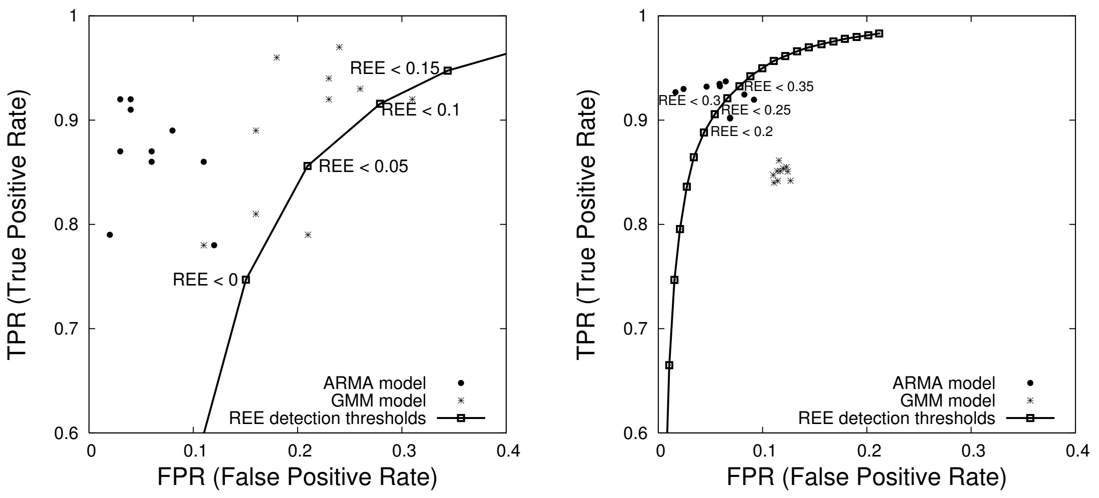
With this second approach, rather than modeling the series of variances, we aim at clustering the REE variances into classes where variances in one cluster are close to each other, and clusters are far apart. In such a way, we would be able to identify the cluster that contains the variance with low mean, and report its elements as variances of the REE of TIV bases. Gaussian Mixture Models (GMMs) [55] are among the most statistically mature methods for clustering, and may be more appropriate than other clustering techniques such as K-means, especially because clusters of variances may have different sizes and correlations between them. As for ARMA models, we do not want to go into unnecessary complex mathematical descriptions and we consider the algorithm as a black box: it takes a variance vector $\vec{v}$ of m entries as input and it builds clusters of near values with these entries. The interested readers can find more details about how the clusters are built in [28].

Although we still need to look for a differentiation between two main clusters (TIV variances and non-TIV variances), we could distinguish more than two clusters, say k. This allows the GMM clustering to create more accurate clusters, from which we choose the cluster that is more likely to contain TIV variances. Let μ_i be the mean value of all values in the cluster i (with $1 \leq i \leq k$). To select the cluster we suspect to contain variances of TIV bases, we look for the cluster corresponding to $\min_i \mu_i$.

4.6.4 Detection results

The plot in figure 4.19 shows the points corresponding to the false positive rates along the x-axis and to the true positive rates along the y-axis, with one point per method used to detect TIV bases, for both data sets with respect to 10 simulation results. By considering ARMA and GMM models we do not need to set any detection threshold. So, for each simulation of Vivaldi, we have only one point (FPR,TPR) in figure 4.19 and the 10 points generated with the 10 simulation results are independent (they do not form a ROC curve). On the other hand, for the detection criterion based on REE, there is one threshold and the points obtained with different values of the threshold for the same simulation results form a ROC curve. For comparison, the ROC curves obtained with different detection thresholds based on REE for one simulation result are presented on figures 4.19(a) and 4.19(b) respectively for the P2PSim and Meridian data sets. We consider different thresholds varying from -1 to 1 by steps of 0.05. The reader should notice that the results obtained for a same value of the threshold are not the same in figure 4.19 and in figure 4.17. This happens because the population on which the REE detection criterion is applied is not the same on both figures: in figure 4.19 we try to detect TIV bases among the paths monitored by Vivaldi (to have something that can be compared to the GMM and ARMA results) while we tried to detect TIV bases among all the paths in figure 4.17.

The first observation on figure 4.19 is that the REE thresholds which give high TPR with low FPR are different for the P2psim and Meridian data sets. The better REE thresholds, when we consider the P2PSim data set (resp. the Meridian data set), vary from 0 to 0.15 (resp. 0.2 to 0.35). In other words, a good REE threshold depends on the used data set, and thus, it will be difficult to fix it a priori.



(a) P2PSim data set (b) Meridian data set

Figure 4.19: Performance of TIV base detection techniques.

We observe that both ARMA and GMM detection methods perform very well for the P2PSim dataset (figure 4.19(a)) comparatively to the ROC curve based on REE detection threshold which is the method proposed in [81]. Note that the ARMA model gives better results than GMM model. Furthemore, the two points that are located at the right of the ROC curve (figure 4.19(a)) represent the detection of TIV bases based on GMM model. The same trend is observed on figure 4.19(b) where all the detection of TIV bases based on GMM model are located on the right-hand side of the ROC curve. Nevertheless, the ARMA model can be considered to be excellent in the case of the Meridian data set. In summary, the detection of TIV bases based on ARMA model gives good performance with up to 85% of TIV bases detected, while suspecting non-TIV bases in rare situations (less than 2%). The reason that ARMA model outperforms the GMM model is probably due to the fact that the REE variances do not follow a gaussian distribution.

However, these results have been computed considering simulations of Vivaldi: in practice the RTTs are not constant and the detection results could be mitigated. Moreover, in ICS applying such detection criteria on the same set continuously can lead to a degradation of the detection performance. Indeed, this criterion is based on the assumption that TIVs are measured in the ICS and that they impact the ICS behaviour. If we use the result of the detection criterion to optimize the selection of neighbors to reduce the impact of TIVs (like Wang *et al.* propose to do in [81]) we will modify the behavior of the ICS. Consequently, the FPR can increase while the TPR can decrease at the same time. To avoid such drawback, one solution is to ignore the results of detection if most node pairs are considered as TIV bases.

4.6.5 Finding a discriminative TIV indicator by supervised learning

In section 4.6.4, we have shown that a TIV base detection based on the REE variance can provide better detection results than a criterion based on the REE. But there exist perhaps more discriminative variables than the REE variance for TIV base detection. Testing all possible variables like we have done with the REE variance would take too much time. To solve this problem, Yongjun Liao proposed to extend our work using supervised learning. She collected training data from Vivaldi, she extracted as many variables of different kinds as possible and she applied a classical supervised learning method (namely, a decision tree) to find the most discriminative variable. Her results have been published in [43] and we will summarize them here.

Variables considered

As in section 4.6.3 simulations of Vivaldi have been run on the P2PSim and the Meridian data sets and we consider the coordinates of the nodes during the last K simulation ticks. We worked essentially with $K = 100$. From the coordinates, we computed the estimated RTTs for the paths between the nodes and their neighbors (as in section 4.6.3 we try only to detect the TIV bases among these paths). For a given path AB, we denote the measured distance by d (*i.e.* $d = RTT(A, B)$) and the estimated distance by $\hat{d}$ (*i.e.* $\hat{d} = EST(A, B)$). Thus, for each path between a node and one of its neighbors, we have $K + 1$ values: d, $\hat{d}_1$, $\hat{d}_2$,..., $\hat{d}_K$. For the K estimated values, we calculated some statistics including

$$\begin{aligned}
\hat{d}_{max} &= max\{\hat{d}_1, \hat{d}_2, \ldots, \hat{d}_K\} \\
\hat{d}_{min} &= min\{\hat{d}_1, \hat{d}_2, \ldots, \hat{d}_K\} \\
\hat{d}_{mean} &= mean\{\hat{d}_1, \hat{d}_2, \ldots, \hat{d}_K\} \\
\hat{d}_{median} &= median\{\hat{d}_1, \hat{d}_2, \ldots, \hat{d}_K\} \\
\hat{d}_{std} &= standard\_deviation\{\hat{d}_1, \hat{d}_2, \ldots, \hat{d}_K\}
\end{aligned}$$

Based on these values, we defined 64 variables on which a detection criterion can be developed. A few examples of these variables are:

$$\frac{\hat{d}_{max} - \hat{d}_{min}}{d} \quad \frac{\hat{d}_{mean} - \hat{d}_{median}}{d} \quad \frac{\hat{d}_{mean} - d}{\hat{d}_{max}} \quad \frac{\hat{d}_{mean}}{\hat{d}_{std}} \quad \frac{\hat{d}_K}{d} \quad \frac{\hat{d}_{std}}{d} \quad \cdots$$

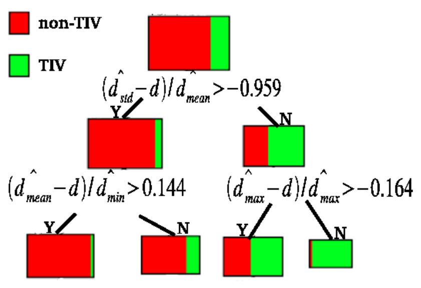
Note that the variable $\hat{d}_K/d$ is equivalent to the variable used by the criterion based on REE (see section 4.6.2) and that the variable $\hat{d}_{std}/d$ is equivalent to the variable used by the criterion based on REE variance (see section 4.6.3).

Supervised learning and Decision Tree

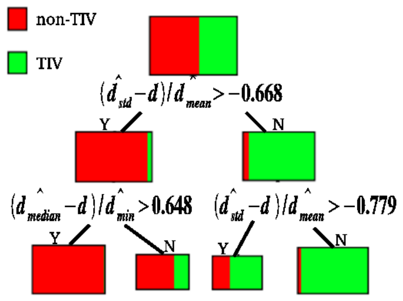
Generally speaking, supervised learning methods take N input/output pairs (x_1, y_1), ..., (x_N, y_N) where x_i is the vector of the input variables and y_i is the output value, and try to reveal the relationships between the inputs and the outputs. In other words, the goal of supervised learning is to learn a function $f(x)$ from a set of training data that predicts, at best, the output y for any new unseen input x. In our case, for each path, we have the variables described above as inputs and a label saying if the path is a TIV base or not as output.

Decision Tree is one of the most popular supervised learning algorithms. Each decision tree is a classifier in the form of a tree structure, where each interior node specifies a binary test carried on a single input variable and each terminal node is labeled with the value of the output. In the learning phase, a decision-tree builder recursively splits the training samples with binary tests, trying to reduce as much as possible the uncertainty about the output classification in the resulting subsets of the samples. The splitting of a node is stopped when the output in it is homogeneous or some other stopping criterion is met. During learning, a byproduct is the ranking of the input variables according to their importance, which is often used to find discriminative variables. In the classification phase, we start from the root of the tree and move through it until a terminal node, where the classification result is provided.

We learned our decision trees using a data mining software called PEPITO [67] which integrates most popular machine learning algorithms. The training data collected through simulations of Vivaldi was randomly divided into disjoint learning and test sets of roughly equal size. In other words, we used half of the data (*i.e.* the variables computed for half of the paths) to build the trees and the other half to evaluate the trees. The trees obtained for the P2PSim and the Meridian data sets are shown in figure 4.20.


🟥 non-TIV
🟩 TIV
$(\hat{d}_{std}-d)/\hat{d}_{mean}>-0.959$
Y
N
$(\hat{d}_{mean}-d)/\hat{d}_{min}>0.144$
$(\hat{d}_{max}-d)/\hat{d}_{max}>-0.164$
Y
N
Y
N

(a) P2PSim data set


🟥 non-TIV
🟩 TIV
$(\hat{d}_{std}-d)/\hat{d}_{mean}>-0.668$
Y
N
$(\hat{d}_{median}-d)/\hat{d}_{min}>0.648$
$(\hat{d}_{std}-d)/\hat{d}_{mean}>-0.779$
Y
N
Y
N

(b) Meridian data set

Figure 4.20: Top three levels of the decision trees built on P2PSim and Meridian data sets. The colour in the rectangular boxes reflects the proportions of TIV and non-TIV bases.

The most discriminative variable: OREE

By examining the root nodes of both trees in figure 4.20, it can be seen that the first test is on the same variable, $(\hat{d}_{std} - d)/\hat{d}_{mean}$, which appears to be the most discriminative. We name this variable OREE because it represents oscillation and relative estimation error: the variable is composed of two parts, $\hat{d}_{std}/\hat{d}_{mean}$ which corresponds to a relative oscillation measure, and $d/\hat{d}_{mean}$ which corresponds to a relative error measure. On both trees, when the value of OREE is smaller than the cut point, the edge is more likely a TIV base, and vice versa.

Figure 4.21 shows the TIV and non-TIV distributions of P2PSim and Meridian data sets with respect to OREE. The first observation is that there is less overlapping between the two curves when we use OREE as variable (see figure 4.21) than when we use REE as variable (see figure 4.16 page 64). Consequently, OREE seems to be a more discriminative variable than REE. It can also be seen that the overlap of the two distributions in P2PSim (figure 4.21(a)) is much larger than in Meridian (figure 4.21(b)), which implies that the TIV bases detection using OREE will probably give better results on the Meridian data set than on the P2PSim data set.

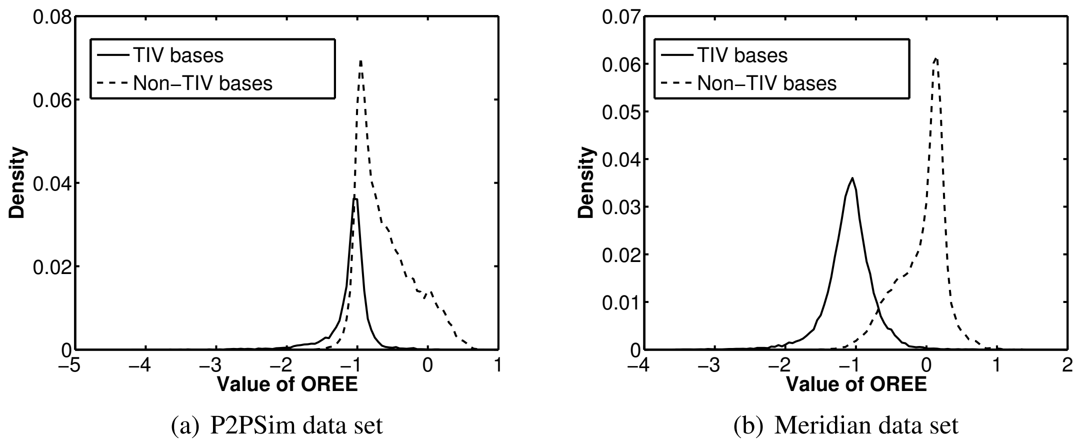


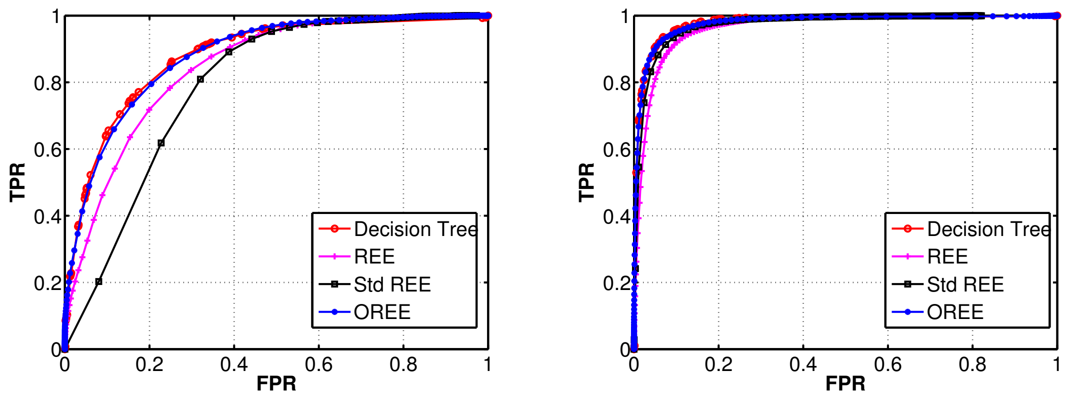
Figure 4.21: TIV and non-TIV bases distributions in P2PSim and Meridian with respect to OREE.

TIV base detection using OREE

A solution to detect TIV bases using OREE is to define thresholds on the OREE value (like we did with the REE variable) and to consider that all the paths which have an OREE smaller than the threshold are TIV bases. Figures 4.22(a) and 4.22(b) compare the ROC curves of OREE, REE and std_REE (*i.e.* the REE variance) on the P2PSim and the Meridian data set respectively. These curves have been obtained by thresholding the values of the corresponding variables.

We see that the ROC curve of OREE is consistently the highest while the other two variables perform inconsistently: REE is more discriminative on P2PSim but less on Meridian than std_REE. Figure 4.22 shows also a ROC curve named "Decision Tree". That curve has been obtained considering the whole detection process described by the decision tree[4] and by varying the threshold of the probability of TIV. Obviously, that curve is better than the "OREE" curve because considering more levels than only the first level (*i.e.* the OREE level) in the decision tree allows to refine the detection results. Nevertheless, on both Meridian and P2PSim data sets, there is only a small difference between the "Decision Tree" curve and the "OREE" curve. This suggests that the other variables provide little extra information so that their corresponding nodes in the trees can be safely pruned with no performance degradation.

[4]For the "OREE" curve we consider only the first node of the decision tree.



(a) P2PSim data set (b) Meridian data set

Figure 4.22: ROC curves for OREE, REE and std_REE obtained by thresholding the values of the corresponding variables. The "Decision Tree" ROC curve has been obtained considering the whole detection process described by the decision tree.

The problem with OREE is that the decision trees in figure 4.20 have non-negligible variations. Even for the two root nodes that correspond to the same variable, a shift is found between the optimal cut points in them, which introduces errors when the threshold for OREE learned on one data set is applied to another. Moreover, even if we can collect data and learn a specific classifier for each network, the TIV distribution in the network often evolves in time due to, for example, changes in routing or network upgrade.

Another possibility to detect TIV bases using OREE is to work like we did with the REE variance: for each node, we can consider the 32 neighboring paths, rank these paths in increasing order following their OREE value and consider that the n first paths in the ranking (*i.e.* the n paths which have the smaller OREE value) are TIV bases. This way, the detection criterion uses a threshold on the value of n rather than a threshold defined directly on the value of OREE. Experiments have revealed that the best choice for the value of the threshold is somewhere around $n = 10$ for the P2PSim data set. With such threshold, the detection criterion provide a TPR around 80% for a FPR smaller than 20%. On the Meridian data set, with the same threshold, we obtain a TPR around 75% and a very small FPR (smaller than 5%). These results are satisfactory for Meridian but this is not the best choice for the threshold on this data set. On Meridian, the best choice for the threshold seems to be somewhere around $n = 15$. This gives a TPR around 98% with a FPR smaller than 10%. But the threshold $n = 15$ is unsuitable for the P2PSim data set because it generates a FPR bigger than 35%. So, it is as difficult to find a threshold on n suitable for all data set than finding a threshold on the value of OREE. However, in practice, it is easier to choose a trade off value for the threshold on n than for the threshold on $OREE$ because (1) n has a fixed range of $[1, 32]$ while the value of OREE is theoretically unbounded and (2) the detection results are influenced by n more evenly than by the value of OREE.

4.7 Conclusion

In this chapter, we have seen that routing shortcuts situations are in fact TIV situations. Such situations are common in the Internet and are a well-known problem for an ICS like Vivaldi. Indeed, situations where three nodes A, B and C are such that

$$RTT(A, B) > RTT(A, C) + RTT(C, B)$$

are not embeddable in a metric space (where the triangle inequality must hold) and will generate unavoidable estimation errors. It has as consequence that the simple solution proposed in the conclusion of the previous chapter to detect routing shortcuts using estimations only is inefficient: since estimations are distances in a metric space, for two given nodes A et B, there exists no node C such that

$$EST(A, B) > EST(A, C) + EST(C, B)$$

Since TIVs are not embeddable in a metric space, they have an impact on the ICS's behaviour. Like Lumezanu *et al.* [48], we tried to detect the TIVs (*i.e.* the routing shortcuts) by observing the behaviour of Vivaldi. However, we did not obtain satisfactory results mainly because the impact of a TIV depends of what is measured by the ICS, and because that impact is also affected by other TIVs. So, it is very difficult to characterize the impact of the TIVs on the ICS in order to propose a criterion to detect them and combining a few measurements with the estimations seems necessary to provide a suitable routing shortcut detection mechanism. We will investigate this in chapter 5.

Even if detecting TIVs using only Vivaldi's estimations seems very difficult, we found several ways to detect TIV bases by observing Vivaldi's behaviour. We saw that a TIV base detection criterion based on REE (proposed in [81]) can provide quite accurate detection results but we showed that a criterion based on the REE variance or on the variable OREE can provide better detection results. A TIV base detection criterion could be useful as a preliminary filter for our overlay routing mechanism: if we are able to directly know that there exists no shortcut for a given path AB, we can avoid testing each potential relay C to know if it is a shortcut or not and, thus, spare resources. But none of the criteria presented in the chapter are suitable for this purpose: criteria on OREE and REE variance are not able to provide an answer for any path AB (they provide answers only for the paths between Vivaldi neighbors) and the criterion on REE is not accurate[5] enough. So, we will not use such filter in chapter 5. However, such criterion is not completely useless. For example, it can be used to improve the selection of neighbors in order to mitigate TIV's impact on the estimations as proposed in [81] (and the criteria presented in this chapter have been developed for this purpose).

[5]For a first filter, we need essentially a criterion which returns always "true" when there exists a routing shortcut (otherwise there will be many cases where we will not search shortcuts while some exist). In other words, we need a very high TPR. The FPR is less critical: if the criterion returns "true" when there exists no shortcut, we will simply waste resources to test each potential shortcut. With a criterion based on REE, to obtain a very high TPR, we need a threshold that will also provide a very high FPR (especially on the P2PSim data set). In other words, such filter will return "true" for almost all the paths and is therefore a useless filter.

Finally, in the light of our observations about the TIVs in the Internet and their impact on Vivaldi, we propose a hierarchical structure of Vivaldi to mitigate the impact of most severe TIVs on the quality of the estimations. The hierarchical approach for Vivaldi will be discussed in chapter 8 of the thesis.

Chapter 5

Routing shortcut detection using an ICS

Abstract

In the previous chapter, we have seen that detecting TIVs using only estimations is impossible because routing shortcuts cannot be embedded in the metric space. Since routing shortcuts generate estimation errors, we tried to detect them by observing the behaviour of the ICS. But we observed that the impact of the TIVs on the ICS is very difficult to characterize and we were only able to suspect the existence of shortcuts for a given path AB without being able to find which node C is a shortcut. In the current chapter, we will combine estimations with a few measurements in order to allow TIV detection. Even if such detection criteria are not able to detect all the shortcuts in the network, we will see that they are at least able to provide an interesting shortcut for lots of paths. Most of the results presented in this chapter were published in [8].

5.1 Combining estimations and measurements

In the previous chapter of this thesis, we have seen that using only the estimations provided by an ICS to find the one-hop shortcuts in a network is impossible: for a given path AB, it is impossible to find a node C such that

$$EST(A,B) > EST(A,C) + EST(C,B) \tag{5.1}$$

So, we have to combine estimations with measurements in order to obtain a shortcut detection criterion. But, our primary objective is to provide a scalable service and we want do as less measurements as possible. With this in mind, we consider that we can obtain the following measurement results:

- First, if we look for a shortcut for the path AB, we assume that $RTT(A,B)$ can be measured. If we search routing shortcuts for a given path AB it consists only of one measurement. What we want absolutely to avoid is to do measurements between A and each potential relay C and between B and each potential relay C. Requiring the measurement of the path for which we search a shortcut becomes problematic only if we want to optimize routes in the whole network. Indeed, if we want to find routing shortcuts for every path of the network, then we have to measure every path in the network, *i.e.* exactly what we want to avoid. However, it has been shown [30] that trying to improve the routing in the whole network can finally lead to a degradation of the performance and we stated in section 2.3 that we do not intend to improve the whole network in order to avoid such problems. Thus, following our objective, requiring the measurement of the direct path is not a problem. Nevertheless, the possibility to detect routing shortcuts for a given path AB without measuring $RTT(A,B)$ will be investigated in section 5.7.

- Secondly, we assume that we can obtain Vivaldi's measurement results between the nodes and their neighbors. These measurements are available and using them will only imply a small additional communication cost to exchange these measurement results between the nodes (in addition to the coordinates).

5.2 Two basic one-hop shortcut detection criteria

For a given path AB, given the measurements described in section 5.1 and estimations, we want to find criteria that provide a set of C nodes that are probably one-hop shortcuts for that path. As such criteria can provide a potentially large set of nodes, we need also a way to rank the C nodes in order to find the best shortcuts as fast as possible.

5.2.1 Detection criteria definitions

In the previous chapter, we observed that more or less 80% of the paths for which there exists at least one significant shortcut are under-estimated by the ICS. The following criteria are based on that observation. Indeed, if the estimation of the alternative path is reliable and if the path AB is significantly under-estimated, we will restore the TIV by replacing $EST(A,B)$ by $RTT(A,B)$ in equation (5.1). This is illustrated in example 6.

Example 6. *Consider the TIV on figure 5.1. The right part of the figure gives a possible embedding for that TIV. Obviously, C does not appear to be a shortcut for AB if we use the estimations: we have $EST(A,C) + EST(C,B) = 75\ ms$, $EST(A,B) = 70\ ms$ and $70 < 75$. Now, consider that we have the measurement result for $RTT(A,B)$. If we replace $EST(A,B)$ by $RTT(A,B)$ in equation (5.1), we have $100 > EST(A,C) + EST(C,B)$ and C appears to be a shortcut for AB (the TIV is restored).*

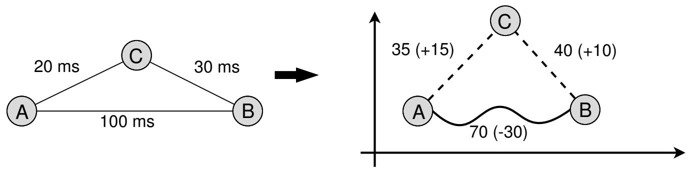


Figure 5.1: **Example of TIV embedding.** If we replace the estimation of the TIV base by its measurement, we restore the TIV and C can be considered as a shortcut for the path AB.

For this example we made the assumption that the short edges of the TIV are over-estimated. In section 4.5, we saw that the short edges are not always over-estimated (the probability of over-estimation for the alternative path is around 50%). However, considering an over-estimation of the alternative path is the worst case in our situation. Indeed, if the alternative path is under-estimated, it appears even shorter that it is and the TIV restoration will be easier.

In example 6, we used the estimations provided by the ICS to approximate the RTT of the alternative path. But, there are other possibilities for that approximation. In this section, we will describe two detection criteria (namely, EDC and ADC) which use different ways to approximate the RTT of the alternative path.

Estimation Detection Criterion (EDC)

EDC is our first criterion. To decide if a node C is a shortcut for a path AB, this criterion compares the measured RTT of the direct path between A and B and the estimated RTT of the alternative path using C as relay (as we did in example 6). Formally, a node C is considered as a shortcut for the path AB if

$$RTT(A,B) > EST(A,C) + EST(C,B) \tag{5.2}$$

The potential problem with that criterion is that it uses the values of the estimations provided by the ICS as if there were no estimation error. However, we know that there are estimation errors and, in particular, that these errors cannot be avoided if node C is a shortcut for the path AB. In example 6, we saw that an under-estimation of the alternative path in not a problem and will allow a detection of C as a shortcut using EDC. Reversely, a too large over-estimation of the alternative path can be problematic and can prevent a detection of C as a shortcut. Anyway, estimation errors may be problematic when we will have to rank the C nodes detected as shortcuts by EDC in order to select the best one. Thus, using directly the values of the estimated RTTs to approximate the RTT of the alternative paths is not necessarily a good idea.

Moreover, in practice, the estimations provided by an ICS are generally not used for their exact values because it is well known that there are unavoidable estimation errors. In most cases, the estimations are used to obtain a rough delay prediction: even if there are estimation errors, the estimations allow a node to estimate if another node is far or near. Using such information, coordinates can be used, for example, to build overlay topologies where overlay links are established between nearly nodes, to find the server that is the nearest to one given node, *etc*. Our second criterion follows that principle and relies only on estimations to decide if a node is near another given node.

Approximation Detection Criterion (ADC)

ADC is our second criterion. For a path AB and a node C, we define C_A (resp. C_B) as C's nearest node among A's (resp. B's) Vivaldi neighbors according to the estimated RTTs. Since A and C_A (resp. B and C_B) are neighbors, we assume that $RTT(A, C_A)$

(resp. $RTT(B, C_B)$) is known and can be used by the criterion to approximate the RTT of the alternative path: a node C is considered as a shortcut for the path AB if,

$$RTT(A, B) > RTT(A, C_A) + RTT(C_B, B) \tag{5.3}$$

5.2.2 Ranking of the detected C nodes

We have two criteria which, for a given path AB, are able to return a set of C nodes that are probably shortcuts for that path. The problem with such criteria is that they do not provide a set of nodes containing only the best shortcuts: they provide a possibly large set of nodes containing nodes that are important shortcuts, nodes that are less important shortcuts and even nodes that are not shortcuts (detection errors). So, we need a way to rank the C nodes of a set in order to find quickly and easily the best shortcuts in that set. Since we want to find the node C providing the smallest RTT for a path between A and B, we will rank the C nodes by order of provided gain. As stated respectively in definitions 15 and 14 page 54, for a path AB, the *absolute gain* (G_a) and the *relative gain* (G_r) provided by a node C are

$$G_a = RTT(A, B) - (RTT(A, C) + RTT(C, B)) \qquad G_r = \frac{G_a}{RTT(A, B)}$$

If C is a shortcut for the path AB, then G_a and G_r will have positive values and the most interesting shortcut is the one that provides the highest value for these parameters. However, we cannot compute G_a and G_r for all C nodes. Indeed, generally, we do not know the real RTT of the alternative path that uses node C: we only have Vivaldi's estimations for that path. As we have used an estimation/approximation for the RTT of the alternative path in the shortcut detection criteria, we will also use that estimation/approximation in the ranking criteria. The values used to rank the C nodes of a set will be denoted *estimated absolute gain* (EG_a) and *estimated relative gain* (EG_r). The definitions of these values depend on the shortcut detection criterion used to obtain the set of C nodes:

$$EG_a = RTT(A, B) - (ERTT(A, C) + ERTT(C, B)) \qquad EG_r = \frac{EG_a}{RTT(A, B)} \tag{5.4}$$

where $ERTT(X, Y)$ is the value used by the detection criterion to estimate the RTT of the path XY.

For a path AB, we will rank the C nodes of the set selected by a shortcut detection criterion in decreasing order of their estimated gain. If the nodes with the highest estimated gains are also those with the highest (real) gains then we will find the nodes providing the most interesting shortcuts in the top of the ranking.

5.3 Performance evaluation

To model Internet latency, we used the three delay matrices described in section 3.4 page 41: P2PSim, Meridian and Planetlab. In these matrices, the percentage of paths for which there exists at least one shortcut is respectively 86%, 97% and 67%. Since a shortcut is not necessary useful[1], we define the notion of *interesting shortcut.*

Definition 16. *Let A, B and C be three nodes. The node C is an* interesting shortcut *for the path AB if it provides at least an absolute gain of $10\,ms$ and a relative gain of 10%.*

The percentage of paths for which there exists at least an interesting shortcut in our matrices is respectively 43%, 83% and 16%. So, searching shortcuts in the networks modelled by these matrices can provide a significant delay improvement for many paths.

We have simulated the behaviour of Vivaldi on these three networks by using the P2PSim [64] discrete-event simulator. As discussed in chapter 3, each node has computed its coordinates in a 10-dimensional Euclidean space by doing measurements with 32 neighbors. Then, we simply applied our detection criteria using the estimated delay matrices computed with the coordinates obtained at the end of the simulations. We will now evaluate the quality of the sets of detected nodes provided by our criteria.

5.3.1 Shortcut detection

To evaluate the performance of our detection criteria, we first use the classical true positive rate and false positive rate indicators. For a path AB, a good shortcut detection criterion must detect a node C as a shortcut if it is a shortcut for the path AB (i.e., if it is a positive) and must reject a node C if it is not a shortcut for the path AB (i.e., if it is a negative). The percentage of positives detected as shortcuts is the *true positive rate* (TPR) and the percentage of negatives detected as shortcuts is the *false positive rate* (FPR). We also define the *interesting true positive rate* (ITPR) as the percentage of interesting shortcuts detected as shortcuts by the criterion. A good detection criterion must provide a high (I)TPR and a low FPR.

	EDC			ADC		
	TPR	ITPR	FPR	TPR	ITPR	FPR
P2PSim	53%	83%	2%	65%	84%	9%
Meridian	54%	64%	9%	70%	76%	25%
Planetlab	37%	75%	1%	60%	81%	5%

Table 5.1: EDC and ADC shortcut detection results

The true positive rates and false positive rates obtained with our criteria are given in table 5.1. We see that the percentage of interesting shortcuts detected as shortcuts (ITPR) is good in most of the cases for both criteria. Furthermore, the percentage of non-shortcuts detected as shortcuts (FPR) is generally quite low. So, these results are satisfactory and,

[1]For example, for a path AB such that $RTT(A,B) = 100\,ms$, a node C such that $RTT(A,C) + RTT(C,B) = 99\,ms$ is a shortcut that provides an absolute gain of $1\,ms$ and a relative gain of 1%. Since using C as relay for sending data from A to B will add an additional forwarding delay, such shortcuts are useless in practice.

considering these results, EDC seems to perform better than ADC: although ADC is always able to detect slightly more shortcuts than EDC, it also gives more false positives.

5.3.2 Detection of the best shortcuts

Being able to detect lots of the shortcuts in a network is one thing, but what matters most is to detect the most interesting shortcuts (those that provide the most important gain). Considering only the paths for which there exists at least one interesting shortcut, the percentage of paths for which the most interesting shortcut is detected in the matrices P2PSim, Meridian and Planetlab is respectively 36%, 41% and 49% with the EDC criterion and 68%, 80% and 70% with the ADC criterion.

Regarding those results ADC seems to be a better criterion than EDC. Indeed, EDC is able to find the best shortcut for 40% of the paths (on average) while the ADC is able to find the best shortcut for more than 70% of the paths in each matrix. However, we must perhaps moderate our conclusion. Firstly because ADC returns large sets of C nodes (including a non-negligible number of false positives) compared to EDC. At the limit, a criterion that would detect as shortcut all C nodes will obviously detect the best shortcut for each path but is completely useless. So, if we choose to use ADC, we absolutely need a criterion[2] to rank the C nodes of a set in order to keep only a subset of the nodes. Moreover, EDC can give better results than we think. Indeed, even if a criterion cannot find the best shortcut for a path, it may be able to find another shortcut that provides almost the same gain. We will investigate that during the evaluation of the quality of the ranking of the C nodes.

5.3.3 Ranking of the detected nodes

For a given matrix and a given detection criterion, we proposed in section 5.2.2 to rank the C nodes of each path on the basis of the EG_r they provide. We will now evaluate whether this gives a ranking with the C nodes providing the best G_r in the first positions and allows us to find easily the best shortcuts among the detected C nodes. For this evaluation, we only consider the paths for which there exists at least one interesting shortcut.

Theoretical correlation computation

A ranking based on estimated values (denoted as estimated ranking in the sequel) is suitable only if it corresponds to the ranking obtained with the real gains (denoted as real ranking in the sequel)In other words, we need a *correlation* between the two rankings.

Definition 17. *In statistics, a rank correlation is the relationship between different rankings of the same set of items. A rank correlation coefficient measures the degree of similarity between two rankings, and can be used to assess its significance. (source: Wikipedia)*

[2]Such criterion can also be useful for EDC because, even if the sets of C nodes are generally smaller than those returned by ADC, they can contain tens or hundreds of nodes.

In statistics, one of the most popular rank correlation coefficients is the *Spearman's rank correlation coefficient* denoted by ρ. Consider that we have two variables X and Y where X_i and Y_i are two values corresponding to a same observation. In our case, X_i and Y_i are the real gain and the estimated gain provided by the node C_i for the path AB. Let x_i (resp. y_i) be the position of the value X_i (resp. Y_i) in the ranking. If $d_i = x_i - y_i$ denotes the difference between the ranks of each observation of the variables (*i.e.* each C_i), then ρ is given by

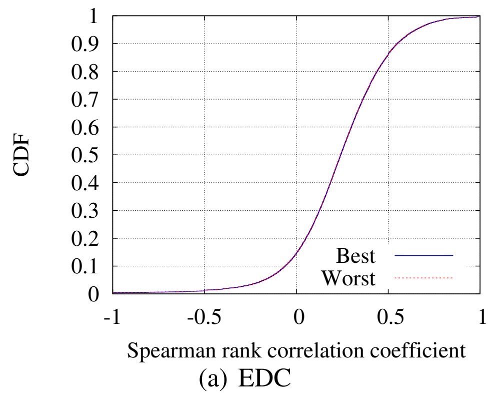
$$\rho = 1 - \frac{6 \times \sum_i d_i^2}{n \times (n^2 - 1)} \tag{5.5}$$

If the ranking based on X and Y are identical, then ρ is equal to 1. More generally, if Y tends to increase when X increases, then ρ is a positive value. If the ranking based on Y is reversed compared to the ranking based on X, then ρ is equal to -1. More generally, if Y tends to decrease when X increase, then ρ is a negative value. If $\rho = 0$, then there is no tendency for Y to increase or decrease when X increases (*i.e.* there is no correlation between the rankings). Note that equation (5.5) is a simplified definition of ρ where it is supposed that there are no ties. But, in our case, we can have ties: two nodes C_i and C_j can provide a same (estimated) gain for a given path AB. There exists a general form of ρ but, in order to avoid to be too optimist or pessimist about the correlation results, we will manage the ties ourselves and we will consider, for each tie, the best case and the worst case from the correlation point of view. Thus, we know that the result obtained in practice will be somewhere between these two values.

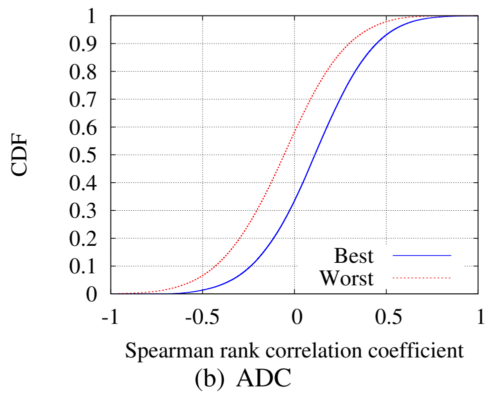
To evaluate the correlation between the two rankings, we considered the C nodes detected as shortcuts by EDC and ADC for each path for which there exists at least one interesting shortcut. Then, we ranked these nodes by decreasing order of real gain on one side (the real ranking) and by decreasing order of estimated gain on the other side (the estimated ranking). This way, for each path, we can compute the Spearman's rank correlation coefficients between these two rankings: one best case coefficient and one worst case coefficient depending on how we manage the ties. Finally, for each data set, we computed the CDFs (the best one and the worst one) of the coefficients obtained for the paths of the data set. Figure 5.2 provides the CDFs obtained using EDC or ADC as criteria on the data set P2PSim (similar results have been obtained for the Meridian and the Planetlab data sets but they are not presented here).

The first observation is that the worst case and the best case cannot be differentiated with EDC. That means that there is only a small number of ties with that criterion. With ADC, we have a significant difference between the "best" curve and the "worst" curve. Having ties with ADC is logical because the alternative paths can only be approximated using a limited number of values. Indeed, by definition of ADC, since each node uses 32 neighbors in Vivaldi, each alternative path is approximated by one value selected in a set of $32 \times 32 = 1024$ values. Thus, the probability to obtain the same approximation for two different C nodes is not negligible.

The second observation is that, considering EDC or ADC, there are almost no paths which have a correlation coefficient smaller than -0.5 or bigger than 0.5. So, in most of the cases there is no correlation (or only a little one) between the real ranking and the estimated ranking. Consequently, ranking the detected nodes on the basis of their esti-



(a) EDC



(b) ADC

Figure 5.2: Rank correlation results for the P2PSim data set.

mated gain does not seem to be a suitable solution to find the best shortcuts. However, we do not require a complete correlation between the two rankings. To be able to detect the best shortcuts, we just need that such shortcuts appear at the beginning of the estimated ranking. In fact, we are not interrested in the ranking of the other detected nodes. Particularly, the end of the estimated ranking can even be completely random. Thus, computing a correlation on the whole ranking does not correspond to our requirements. But, to our knowledge, there is no theoretical correlation computation that can represent our particular needs.

Nevertheless, we have compared manually the two rankings obtained for many paths and, most of the time, the best shortcuts with respect to the real gains appear in the first part of the estimated ranking. In the light of these observations, we will evaluate the quality of the ranking of the C nodes by considering that only the first k C nodes of the estimated ranking are detected by the criterion (for several values of the parameter k). This way, if a small value of k allows us to find good shortcuts most of the time, we can consider that the estimated ranking is suitable.

Empirical evaluation

As discussed above, to evaluate the quality of the ranking, we consider for each path that only the first k C nodes of the ranking are detected by the criterion (for several values of the parameter k). For these subsets of C nodes we compute the difference between the G_r provided by the best existing shortcut (denoted by $G_{best}(A,B)$) and the G_r provided by the best shortcut in the subset (denoted by $G_{det}(A,B)$). If no shortcut is detected for the path AB, we define $G_{det}(A,B) = 0$. We thus obtain one value

$$Difference(A,B) = G_{best}(A,B) - G_{det}(A,B)$$

for each path AB and we build the CDF of these values. The CDFs obtained with different values of the parameter k are given in figure 5.3.

The curves named "no detection" in figures 5.3(a) and 5.3(b) give the CDF of the G_r provided by the best existing shortcut for each path of the matrix. Indeed, if there is no

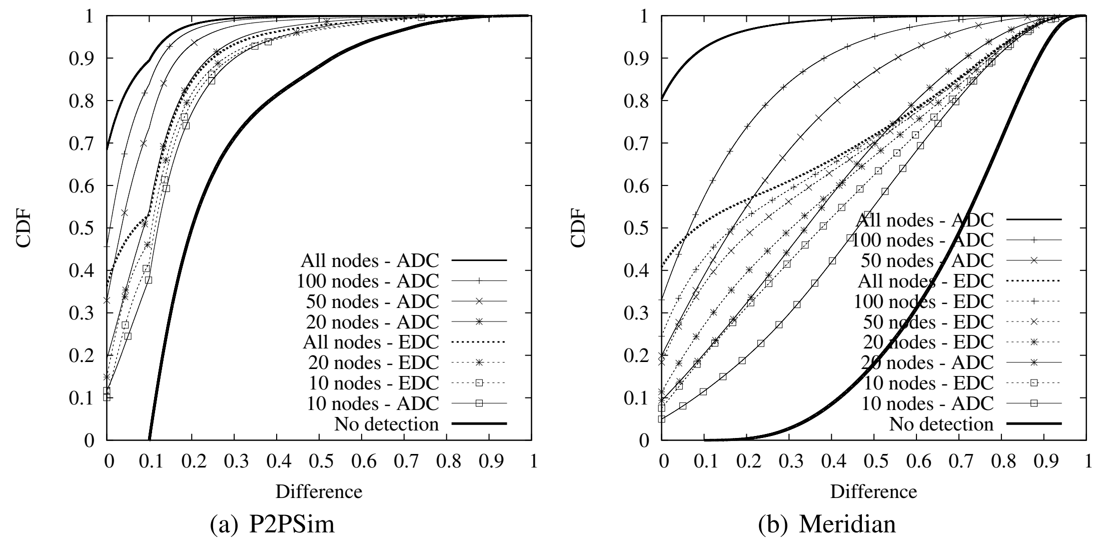


Figure 5.3: Comparison of EDC and ADC. The figure shows the difference of G_r between the best shortcut and the best detected shortcut.

detection criterion applied, there is no shortcut detected and the difference between the G_r provided by the best existing shortcut and the G_r provided by the best detected shortcut is the G_r provided by the best existing shortcut. In other words, these curves represent the potential improvement in the corresponding data set: by applying one shortcut detection criterion, we will detect some shortcuts and, thus, reduce that difference (*i.e.* the potential improvement) for some paths. Since the computed difference is smaller for more paths, the CDF will rise faster on the graphs.

The curves named "all nodes - XDC" in the subfigures of figure 5.3 give the CDF computed by considering all the nodes selected by the shortcut detection criterion XDC (ADC or EDC). This is equivalent to using $k = \infty$. These are the best results that the given detection criterion applied on the given matrix can provide. We can see that ADC still gives better results than EDC considering those graphs. Indeed, with ADC, there are only a small part of the paths for which the difference between the G_r provided by the best existing shortcut and the G_r provided by the best detected shortcut is bigger than 0.2. That means that, for a small part of the paths AB, it is still possible to find another shortcut C that would improve by more than 20% of $RTT(A, B)$ the gain provided by the alternative path proposed by our shortcut detection criterion. This difference is generally bigger with EDC.

Let us see the situation if we keep only the first nodes of the rankings. The graphs named "k nodes - XDC" in the subfigures of figure 5.3 give the CDF computed by considering as detected only the first k nodes of the rankings obtained by using the shortcut detection criterion XDC (ADC or EDC). The first thing we see is that even if we take only a few nodes in the ranking (e.g., 10 nodes), we obtain already a good improvement compared to the situation whithout shortcut detection. We also see that ADC gives better results than EDC only if we keep a sufficient number of nodes: more than 50 nodes for Meridian, more than 20 nodes for P2PSim and more than 5 nodes for Planetlab[3]. Moreover, if we keep a sufficient number of nodes (100 nodes for Meridian, 20 nodes for P2PSim and 10 nodes for Planetlab), we obtain a result with ADC that is better than what we can obtain by considering all the nodes with EDC. The number of nodes to keep to obtain good results may seem important for Meridian but it represents only 4% of the total number of nodes.

Given those results we can conclude that, with ADC, when considering only 5% of the total number of nodes in each matrix (that represents 125 nodes for Meridian, 87 nodes for P2PSim and 9 nodes for Planetlab), we are able to provide a significant improvement of the RTT for lots of paths for which there exists at least one interesting shortcut.

5.4 Hybrid one-hop shortcut detection criterion

It is possible to obtain better results than those obtained with ADC. Indeed, if it is impossible to find some A's (resp. B's) Vivaldi neighbors near C, the approximation of $RTT(A,C)$ (resp. $RTT(C,B)$) by $RTT(A,C_A)$ (resp. $RTT(C_B,B)$) can be very bad. In such case, using the EDC criterion can provide more reliable detection results even if there are estimation errors. So, we define a *Hybrid Detection Criterion* (HDC) by combining our two basic criteria in order to exploit their advantages.

5.4.1 Criterion definition

Formally, let C_A (resp. C_B) be C's nearest node among A's (resp. B's) Vivaldi neighbors according to the estimated RTTs. We define

$$\begin{aligned} ERTT(A,C) &= \begin{cases} RTT(A,C_A) & \text{if } EST(C_A,C) < \mathit{threshold} \\ EST(A,C) & \text{otherwise} \end{cases} \\ ERTT(C,B) &= \begin{cases} RTT(C_B,B) & \text{if } EST(C_B,C) < \mathit{threshold} \\ EST(C,B) & \text{otherwise} \end{cases} \end{aligned}$$

where *threshold* is a value used to decide if C_A (resp. C_B) is sufficiently near C to obtain a quite good approximation of $RTT(A,C)$ (resp. $RTT(C,B)$) by using $RTT(A,C_A)$ (resp. $RTT(C_B,B)$). To test the HDC criterion, we choose *threshold* equal to 10% of $RTT(A,B)$ when we search a shortcut for the path AB. A fine tuning of this parameter must still be done but some small experiments have shown that 10% seems a good choice compared to other values. Using these definitions, a node C is considered as a shortcut for the path AB if

$$RTT(A,B) > ERTT(A,C) + ERTT(C,B) \tag{5.6}$$

To rank the C nodes of the set provided by this criterion for a given path AB, we proceed as described in section 5.2.2.

[3]Since there are only 180 nodes in the Planetlab matrix, the sets of C nodes returned by the criteria are quite small and keeping all the detected nodes is not really a problem. So, the quality of the ranking is less important for that matrix and we will not show the graphs here.

5.4.2 Performance evaluation

For the evaluation of HDC, we consider again only the paths for which there exists at least one interesting shortcut. Let us begin with the detection of the best shortcuts. The percentages of paths for which the most interesting shortcut is detected with the HDC in the matrices P2PSim, Meridian and Planetlab are respectively 64%, 71% and 74%. These results are better than those obtained with EDC but are worse than those obtained with ADC. So, HDC misses interesting shortcuts that ADC is able to find. Moreover, in figure 5.4, considering all the nodes detected by the criterion, we can see that ADC is potentially able to provide a better improvement of the latencies in the network than HDC.

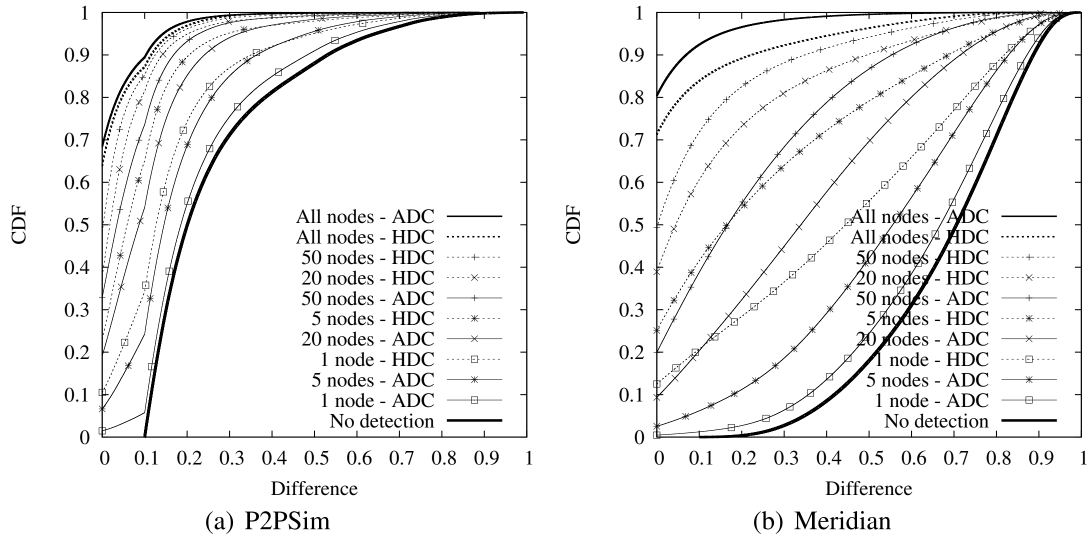


Figure 5.4: Comparison of ADC and HDC: Difference of G_r between the best and the best detected shortcut.

However, considering the quality of the ranking of the C nodes, we see in figure 5.4 (graphs named "k nodes - XDC") that HDC performs a lot better than ADC. Indeed, with HDC, even if we take only the first node of the ranking, we obtain already a significant improvement compared to the situation without shortcut detection. Furthermore, if we only consider the first 5 nodes of a ranking, we obtain better results than if we consider the first 50 nodes of the rankings with ADC. So, with HDC, we can provide a substantial improvement of the RTT for lots of paths by considering only about 2% of the total number of nodes (that represents only 50 nodes for Meridian, 34 nodes for P2PSim and 3 nodes for Planetlab).

5.5 Summary of the detection process

In summary, to find a good shortcut for a given path AB we will proceed as indicated in algorithm 3. The computations in lines 5 and 6 depend of the detection criterion applied.

With EDC, we have:

$$ERTT(A,C) = EST(A,C) \qquad\qquad ERTT(C,B) = EST(C,B)$$

If C_A (resp. C_B) is C's nearest node among A's (resp. B's) Vivaldi neighbors according to the estimated RTTs, with ADC, we have:

$$ERTT(A,C) = RTT(A,C_A) \qquad\qquad ERTT(C,B) = RTT(C_B,B)$$

Finally, with HDC, we have:

$$\begin{aligned} ERTT(A,C) &= \begin{cases} RTT(A,C_A) & \text{if } EST(C_A,C) < \textit{threshold} \\ EST(A,C) & \text{otherwise} \end{cases} \\ ERTT(C,B) &= \begin{cases} RTT(C_B,B) & \text{if } EST(C_B,C) < \textit{threshold} \\ EST(C,B) & \text{otherwise} \end{cases} \end{aligned}$$

Between line 1 and line 12, we use one of our criteria (EDC, ADC or HDC) to build a set $\mathcal{S}$ of candidates for the path AB. Thus, $\mathcal{S}$ contains the C nodes detected as shortcuts by the criterion. If $\mathcal{S}$ contains no node, we stop the process at line 16 by returning B to the application that searches a shortcut or the path AB. Otherwise, through the operations at lines 18 and 19, we keep in a set $\mathcal{S}_k$ the k candidates providing the most important gain with respect to the estimations. Next, between line 22 and line 28, we check these k candidates with measurements in order to find the best one among them. Let C be the best shortcut among the k candidates. If C is really a shortcut, then the algorithm returns C. Otherwise (*i.e.* if C is a false positive), the algorithm returns B. An implementation of this shortcut detection mechanism has been realized by Pierre Lepropre during his master's thesis [38].

Notice that the value of k must be as small as possible because the algorithm requires $2 \times k$ RTT measurements at line 23. Following the results given in section 5.4.2, the value of k depends of the network's size. With HDC, we have seen that considering about 2% of the total number of nodes allows a substantial improvement of the RTT for lots of paths. That corresponds to $k = 50$ for Meridian, $k = 34$ for P2PSim and $k = 3$ for Planetlab.

5.6 Comparison to a random relay choice

Our criteria allow to find good alternative paths for lots of paths in our data sets but they are not perfect. Indeed, they are not able to find the best shortcut for any given path in the network while RON and its measurement-based variants guarantee to find these best shortcuts. Facing a measurement-based approach, even if we are less efficient, we can argue that our estimation based approach is a lot more scalable. But, are we more efficient than a simple shortcut mechanism based on a random choice of relay nodes like the one proposed by Gummadi *et al.* [22]? Since such approach is more scalable than ours, we must be significantly more efficient. We will check whether it is the case in this section.

Algorithm 3 Generic algorithm for finding one shortcut for a given path AB

```
Require: $\mathcal{N}$: set containing all nodes; $A$: path's source; $B$: path's destination; $k$: ranking threshold.
Ensure: Returns the best detected shortcut or $B$ otherwise.
 1: Measure $RTT(A,B)$
 2: $\mathcal{S} = \emptyset$
 3: for all $C \in \mathcal{N}$ do
 4:   if $((C \neq A) \wedge (C \neq B))$ then
 5:     Compute a suitable approximation $ERTT(A,C)$ for $RTT(A,C)$
 6:     Compute a suitable approximation $ERTT(C,B)$ for $RTT(C,B)$
 7:     if $RTT(A,B) > ERTT(A,C) + ERTT(C,B)$ then
 8:       $EG_a = RTT(A,B) - (ERTT(A,C) + ERTT(C,B))$
 9:       $\mathcal{S} = \mathcal{S} \cup (C, EG_a)$
10:     end if
11:   end if
12: end for
13: {$\mathcal{S}$ contains the nodes detected as shortcuts by the criterion (i.e. the candidates).}
14: if $\mathcal{S} == \emptyset$ then
15:   {The criterion applied provides no candidate for the path $AB$.}
16:   return $B$
17: end if
18: Sort the elements of $\mathcal{S}$ by decreasing order of $EG_a$
19: Keep the first $k$ nodes of the ranking in a set $\mathcal{S}_k$
20: {$\mathcal{S}_k$ contains the best $k$ candidates with respect to the estimations.}
21: $\mathcal{S}' = \emptyset$
22: for all $(C, EG_a) \in \mathcal{S}_k$ do
23:   Measure $RTT(A,C)$ and $RTT(C,B)$
24:   $G_a = RTT(A,B) - (RTT(A,C) + RTT(C,B))$
25:   $\mathcal{S}' = \mathcal{S}' \cup (C, G_a)$
26: end for
27: Sort the elements of $\mathcal{S}'$ by decreasing order of $G_a$
28: Let $(C, G_a)$ be the first element of the ranking
29: {$C$ is the best shortcut among the $k$ candidates.}
30: if $G_a > 0$ then
31:   {The best candidate is a true positive.}
32:   return $C$
33: else
34:   {All the candidates are false positives.}
35:   return $B$
36: end if
```

5.6.1 Methodology

Our objective here is to compare the detection results obtained using our best criterion (HDC) when we consider only the k first nodes of the ranking to a random "detection" criterion (RDC). RDC simply selects randomly k nodes as potential shortcuts. In other words, to obtain RDC, we replace the operations between lines 3 and 12 in algorithm 3 by a random selection of k nodes. To evaluate the quality of the detection results we will proceed like in figures 5.3 and 5.4. For each path for which there exists at least one interesting shortcut, we will compute the difference between the gain provided by best existing shortcut and the gain provided by the best detected shortcut (*i.e.* the potential improvement remaining after detection). Then, we will compute the CDF of the values obtained for all the paths in the considered data set. Remember that the faster the obtained CDF rises, the better it is because it indicates that, for lots of paths, there is only a small potential improvement remaining after the detection process.

As for the previous figures, we will consider the estimations obtained with one simulation of Vivaldi. The difference is that RDC is not a deterministic process: applying it multiple times with the same estimations as inputs can produce different detection results. Thus, for each path, we will make 100 random selections of k nodes (for different values of the parameter k) as potential shortcuts. Then, for each path, we will compute the mean of the 100 values obtained by computing the potential improvement remaining after detection for each selection. The curves presenting the detection results obtained with RDC are the CDFs of these mean values. The curves presented in figures 5.3 and 5.4 can be compared to the "average" curves computed for RDC.

5.6.2 Results

The CDFs obtained with different values of the parameter k are given in figure 5.5. The curves named "k nodes - HDC" in the subfigures of figure 5.5 give the CDF computed by considering as detected only the first k nodes of the rankings obtained with HDC. The curves named "k nodes - RDC" give the CDF of the means computed by considering k randomly selected nodes as detected. Recall that the "no detection" curves represent the potential improvement of the corresponding data set before any shortcut detection (*i.e.* the CDF of the G_r provided by the best existing shortcut for each path).

Let us start with P2PSim. We see in figure 5.5(a) that, on average, it is necessary to use a bigger value of k with RDC than with HDC. For example, even if we select randomly 100 nodes with RDC, the improvement of the RTTs is, on average, worse than what we obtain when we consider only the 5 first nodes of the ranking with HDC. Moreover, considering only the first node of the ranking with HDC provides an improvement similar to what we obtain when we select 20 nodes with RDC. This is not shown in figure 5.5(a) but to reach (on average) the same improvement of the RTTs than what we obtain with HDC and $k = 20$, it is necessary to select randomly 500 nodes with RDC. Since HDC and RDC both require $2 \times k$ measurements to discover which is the best shortcut among the k selected nodes, we can say that HDC performs a lot better than RDC on P2PSim. Similar conclusion are obtained for Meridian and Planetlab. For example, with Meridian, if we use RDC with $k = 100$, figure 5.5(b) shows that only about 50% of the path have

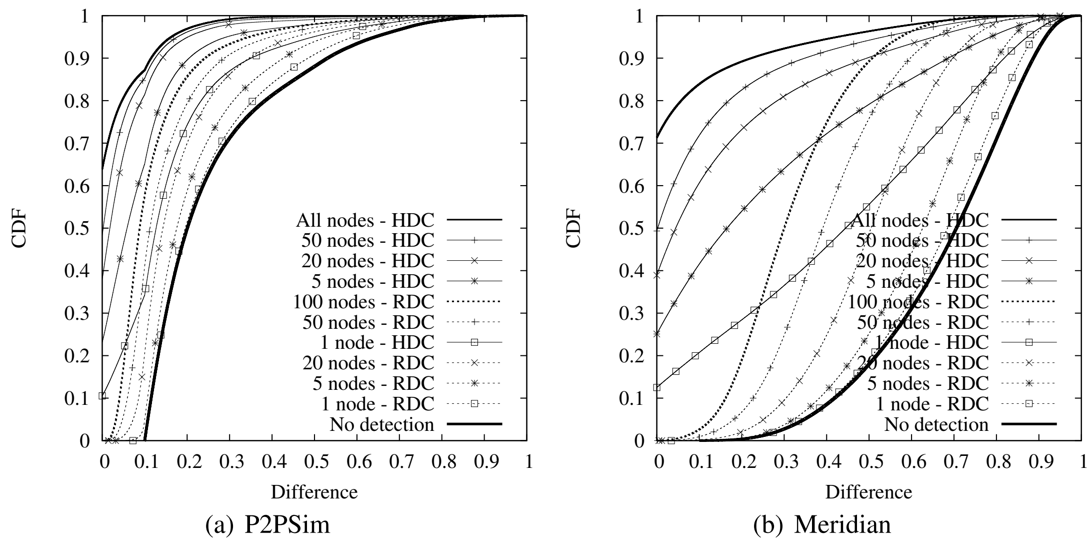


Figure 5.5: Comparison of HDC and RDC: Difference of G_r between the best and the best detected shortcut.

(on average) a potential improvement remaining after detection which is smaller than 30%. For comparison, with HDC and $k = 50$, about 85% of the paths have a potential improvement remaining after detection which is smaller than 30%.

In conclusion, even if a random choice of potential relay nodes is more scalable than our estimation-based approach, HDC allows us to focus on a very small number of C nodes that are probably shortcuts. So, HDC requires significantly less measurements than RDC to find an interesting shortcut for a given path AB: in order to obtain similar results with HDC and RDC, it is necessary to consider respectively about 2% and 30% of the total number of nodes as potential shortcuts. Nevertheless, HDC implies a measurement cost $O(n \times m)$ and a communication cost $O(n^2)$ (that can probably be smaller) to run Vivaldi and exchange coordinates among the nodes. Even if RDC does not induce similar costs, two arguments play in favour of HDC:

- Vivaldi's measurement and communication costs are constant costs: if the detection mechanism is frequently used, these costs are distributed among the usages of the mechanism and can become negligible compared to the measurements costs $O(k)$ generated by each shortcut search. So, if the detection mechanism is frequently used, the estimation-based approach can finally consume less resources than RDC.
- Vivaldi is a generic tool that provides estimations for any path in the network. It is independent from our detection mechanism and its estimations can be used for other purposes. So Vivaldi's measurement and communication costs can be distributed among applications that use it.

In the light of these results we believe that the estimation-based approach is a good trade-off between a simple random choice of relay nodes (which is scalable with a small value of k but also, by definition, completely random in terms of quality of the results) and a measurement-based approach (which guarantees to find the best shortcut for any given path but is absolutely not scalable).

5.7 Reducing the requirement of measurements

The process given in section 5.5 takes some time before providing an answer: if A wants to find a shortcut between him and B, he must measure $RTT(A, B)$, get Vivaldi's measurement results from B, check each potential relay C, keep the best k C nodes with respect to the approximations and, finally, perform measurements to find the best shortcut among these k nodes. All of this cannot be done instantaneously. Since we want to answer as fast as possible to an application that requests a shortcut for a given path AB, making a systematic anticipative shortcut search for each path could be a good idea. However, due to the measurement done at line 1 in algorithm 3, making an anticipative shortcut search for each path in the network using one of our criteria implies measuring the RTT for each path in the network, *i.e.* exactly what we want to avoid. In this section, we will investigate if it is possible to find shortcuts for a given path without measuring that path.

For a given path AB, $RTT(A, B)$ is used by our criteria to decide if a given C node is a shortcut or not for AB and to compute the estimated gains in order to rank the C nodes. The first thing to understand is that $RTT(A, B)$ is not necessary to rank the C nodes. Indeed, by definition, the estimated gain for a given C node is computed as the difference between an identical value for all the C nodes (*i.e.* $RTT(A, B)$) and the approximation of the one-hop path ACB. Consequently, the first node of the ranking with respect to the estimated gain is the C node for which the path ACB is the shortest with respect to the approximations, *etc*. Thus, we can rank the C nodes even if $RTT(A, B)$ is unknown. Nevertheless, $RTT(A, B)$ seems still useful to decide if a given node C is a candidate or not for AB. But, is it necessary to do such preselection? A solution is to rank the C nodes by increasing order of alternative path $ERTT$ and to keep the first k nodes of this ranking. We already know that the order of the ranking will be the same as with our criteria. The only difference is that we will systematically consider k candidates while our criteria consider *at most* k candidates. That can be a bad thing because the next phase of the process (*i.e.* doing measurements to test the candidates) can consume more resources: for each shortcut detection, we will have to do measurements to test exactly k candidates instead of having to do measurements to test at most k candidates. On the other hand, this can be a good thing because we can potentially detect more shortcuts: checking if the approximation of the path ACB is smaller than $RTT(A, B)$ can lead to false negatives[4] that will not appear anymore if we skip that comparison.

5.7.1 Definition of the criteria without measuring $RTT(A, B)$

In this section, we define formally the "2.0" versions of our criteria where we do not use $RTT(A, B)$ to detect shortcuts for the path AB. We simply named these versions of our criteria EDC2, ADC2 and HDC2.

[4]Considering that a C node is not a shortcut when it is.

General detection process

The detection process is adapted from algorithm 3 and is given in algorithm 4. Since the algorithm described here is common to all our criteria, the step where the approximation of the alternative path is computed remains general and will be detailed later for every criteria.

Algorithm 4 Finding one shortcut for a path AB without $RTT(A,B)$ measurement.

```
Require: $\mathcal{N}$: set containing all nodes; $A$: path's source; $B$: path's destination; $k$: ranking threshold.
Ensure: Returns the best candidate (no guarantee that it is a shortcut).
 1: $\mathcal{S} = \emptyset$
 2: for all $C \in \mathcal{N}$ do
 3:     if $((C \neq A) \wedge (C \neq B))$ then
 4:         Compute a suitable approximation $ERTT(A,C,B)$ for the path $ACB$
 5:         $\mathcal{S} = \mathcal{S} \cup (C, ERTT(A,C,B))$
 6:     end if
 7: end for
 8: Sort the elements of $\mathcal{S}$ by increasing order of $ERTT(A,C,B)$
 9: Keep the first $k$ nodes of the ranking in a set $\mathcal{S}_k$
10: {$\mathcal{S}_k$ contains the best $k$ candidates with respect to the estimations.}
11: $\mathcal{S}' = \emptyset$
12: for all $(C, ERTT(A,C,B)) \in \mathcal{S}_k$ do
13:     Measure $RTT(A,C)$ and $RTT(C,B)$
14:     $RTT(A,C,B) = RTT(A,C) + RTT(C,B)$
15:     $\mathcal{S}' = \mathcal{S}' \cup (C, RTT(A,C,B))$
16: end for
17: Sort the elements of $\mathcal{S}'$ by increasing order of $RTT(A,C,B)$
18: Let $(C, RTT(A,C,B))$ be the first element of the ranking
19: {$C$ is the best relay among the $k$ candidates.}
20: return $C$
```

Among the k candidates, the C returned by algorithm 4 is the node that provides the shortest alternative path with respect to the measured RTTs but there is no guarantee that this node is effectively a shortcut. This is not really a problem since our objective here is to perform an anticipative shortcut search. If an application requests a shortcut for the path AB, it is still possible to measure $RTT(A,B)$ at this time and, depending of this measurement result, returning either C, or B. We will do that when we will evaluate the performance of this algorithm. The reader should also notice that algorithm 4 still requires measurements at line 13. These measurements are annoying because they are performed during an anticipative search done for each path. However, the anticipative search phase can be limited to the construction of $\mathcal{S}_k$. Like checking if C is really a shortcut, finding the best candidate among $\mathcal{S}_k$'s elements can be done for the path AB only when a shortcut is requested for that path.

Approximation of ACB in EDC2

As with EDC, we simply use the estimations provided by Vivaldi to approximate the RTT of the alternative path with EDC2. Formally, we have:

$$ERTT(A, C, B) = EST(A, C) + EST(C, B)$$

Approximation of ACB in ADC2

For ADC2, we will also use the same approximation than ADC for the RTT of the alternative path. Formally, if C_A (resp. C_B) is C's nearest node among A's (resp. B's) Vivaldi neighbors according to the estimated RTTs, then

$$ERTT(A, C, B) = RTT(A, C_A) + RTT(C_B, B)$$

Approximation of ACB in HDC2

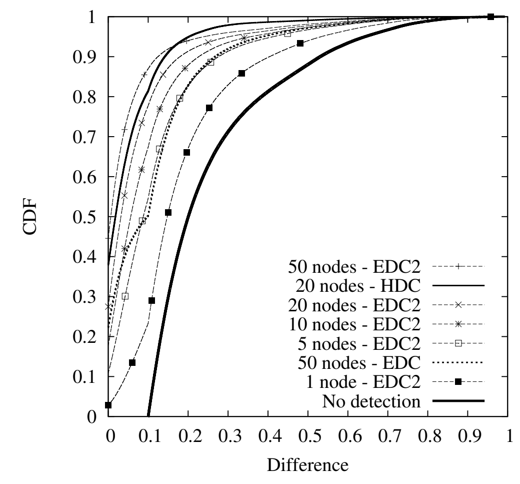
For HDC2, it is a little bit more complicated. Indeed, in HDC, we defined $threshold$ as a percentage of $RTT(A, B)$ when we search a shortcut for the path AB. Since $RTT(A, B)$ is unknown in HDC2, we must use another definition for $threshold$. A simple solution is to define it as a percentage of $EST(A, B)$. For our tests, we choose $threshold$ equal to 10% of $EST(A, B)$. Apart from the modification of the threshold, the RTT of the alternative path is approximated in HDC2 as it is in HDC. Formally, if C_A (resp. C_B) is C's nearest node among A's (resp. B's) Vivaldi neighbors according to the estimated RTTs, then

$$\begin{aligned} ERTT(A, C) &= \begin{cases} RTT(A, C_A) & \text{if } EST(C_A, C) < threshold \\ EST(A, C) & \text{otherwise} \end{cases} \\ ERTT(C, B) &= \begin{cases} RTT(C_B, B) & \text{if } EST(C_B, C) < threshold \\ EST(C, B) & \text{otherwise} \end{cases} \\ ERTT(A, C, B) &= ERTT(A, C) + ERTT(C, B) \end{aligned}$$

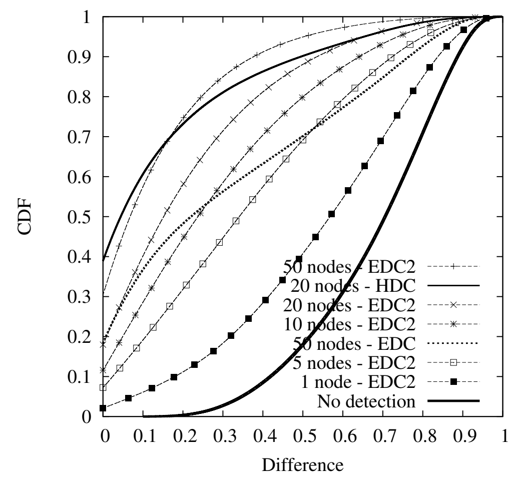
5.7.2 Quality of the detection

In this section, we will consider the ability of these new criteria to find good shortcuts in networks by computing the potential improvement remaining after the detection process. For the P2PSim data set (resp. the Meridian data set), figure 5.6(a) (resp. figure 5.6(b)) shows the results obtained with EDC2, figure 5.6(c) (resp. figure 5.6(d)) shows the results obtained with ADC2 and figure 5.6(e) (resp. figure 5.6(f)) shows the results obtained with HDC2. As usually for this type of figure, we considered only the paths for which there exists at least one interesting shortcut.

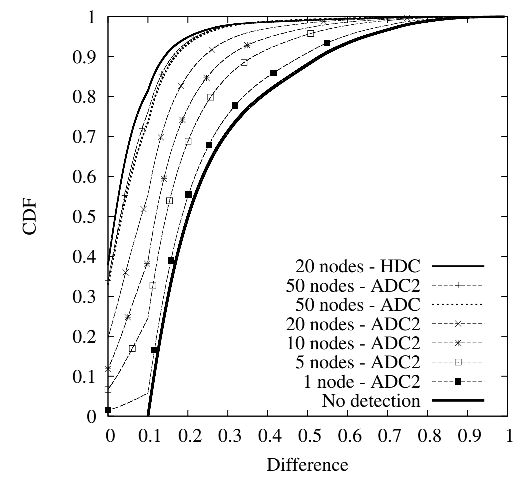
For EDC2, the first observation is that it significantly improves the detection results compared to EDC: on P2PSim (figure 5.6(a)) and Meridian (figure 5.6(b)) we see that the potential improvement remaining after the detection process is smaller when EDC2 is applied with $k = 50$ than when EDC is applied with $k = 50$. On P2PSim, EDC2 applied



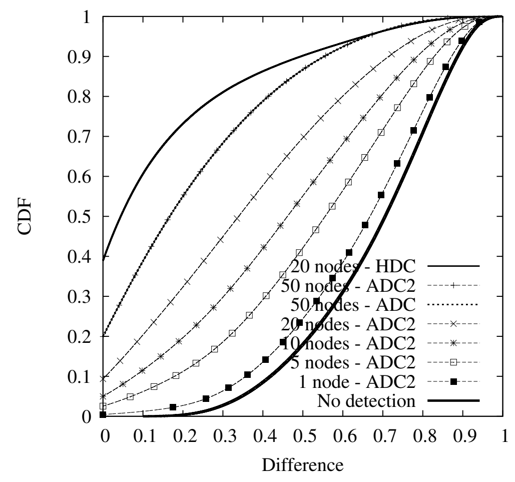
(a) EDC2 - P2PSim



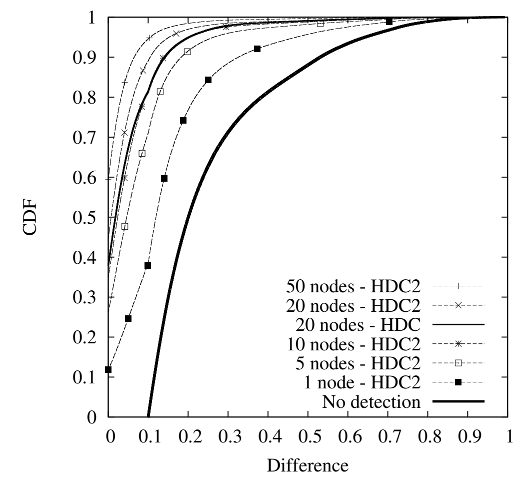
(b) EDC2 - Meridian



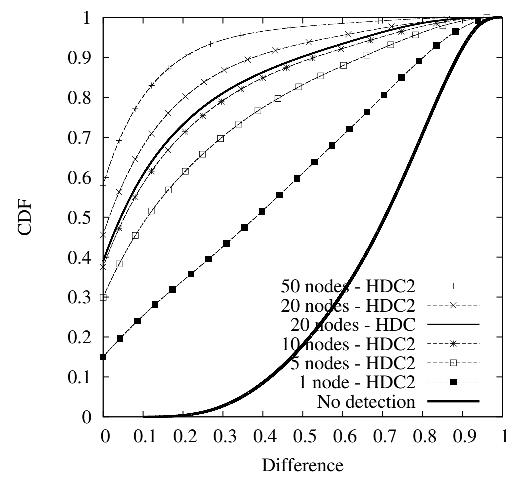
(c) ADC2 - P2PSim



(d) ADC2 - Meridian



(e) HDC2 - P2PSim



(f) HDC2 - Meridian

Figure 5.6: Difference of G_r between the best existing shortcut and the best detected shortcut with the criteria EDC2, ADC2 and HDC2.

with $k = 5$ provides results similar to what we obtain when EDC is applied with $k = 50$. On both data sets, the second observation is that EDC2 applied with $k = 50$ gives almost the same results as HDC applied with $k = 20$. Thus, HDC remains our best one-hop shortcut detection criterion.

For ADC2, we do not observe a significant improvement compared to the results obtained with ADC: in P2PSim (figure 5.6(c)) and Meridian (figure 5.6(d)), we see that the CDFs corresponding to ADC2 applied with $k = 50$ and ADC applied with $k = 50$ are very similar. On P2PSim $ADC2$ applied with $k = 50$ performs a little bit better than ADC applied with $k = 50$ but the improvement is negligible.

For HDC2, figures 5.6(e) and 5.6(f) show that HDC2 provide better detection results than HDC on both data sets: when HDC2 is applied with $k = 20$, the potential improvement remaining after de detection process is significantly smaller than when HDC is applied with $k = 20$. On both data sets the detection results obtained when HDC is applied with $k = 20$ (our reference criterion) are similar to those obtained when HDC2 is applied with $k = 10$. In the light of this observation, HDC2 seems to be our best detection criterion. However, we know that, for a same value of the parameter k, HDC2 will perform more measurements than HDC: for each shortcut detection, HDC2 will have to do measurements to test exactly k candidates while HDC has to do measurements to test at most k candidates. In section 5.7.3 we will evaluate if applying HDC2 with $k = 10$ generates more measurements than applying HDC with $k = 20$.

5.7.3 Additional measurements

Being able to improve the RTT is one thing but we have already discussed the fact that our new criteria can finally generate more measurements than our first criteria. If these additional measurements allow to find more (interesting) shortcuts (*i.e.* if they increase the (I)TPR), then they can be considered as useful[5] and these additional measurements are problematic only if they significantly increase the false positive rate (*i.e.* if they are useless measurements). Particulary, this will be the case for the paths that are not TIV bases. For such paths EDC, ADC and HDC can provide a very small (or an empty) list of candidates, and can do only a small number of measurements (or no measurement) to check if they are shortcuts or not. With EDC2, ADC2 and HDC2, k measurements will systematically be done to discover that there exists no shortcut.

To evaluate if this problem is negligible or not, we will compute the FPR obtained with the criteria. Note that the values computed here for EDC, ADC and HDC are different from those computed in section 5.3.1. In section 5.3.1, we considered all the elements from $\mathcal{S}$ as positives (*i.e.* $k = \infty$). Here we will compute the FPR obtained with different values of the parameter k and only the first k C nodes in the ranking are considered as

[5]Indeed, even if the detection process as it is presented here returns only the best shortcut detected, we can imagine to return an ordered list of shortcuts. This way, the first element in the returned list was the best one during the detection process and the application who asked a shortcut has available backup relay nodes.

positives[6]. The FPRs obtained with different values of k are given table 5.2 for P2PSim and in table 5.3 for Meridian. Considering that we keep the first k positives only, the (I)TPRs are not relevant any longer. Thus, (I)TPRs are not presented in the tables.

	$k = 1$	$k = 5$	$k = 10$	$k = 20$	$k = 50$	$k = \infty$
EDC	0.01	0.06	0.1	0.2	0.4	2.2
EDC2	0.04	0.2	0.5	1.0	2.5	—
ADC	0.06	0.3	0.5	1.0	2.2	9.2
ADC2	0.06	0.3	0.6	1.1	2.7	—
HDC	0.02	0.1	0.2	0.4	0.8	2.9
HDC2	0.04	0.2	0.4	0.9	2.4	—

Table 5.2: FPRs obtained for P2PSim with different values of k and different criteria (in $\%$)

	$k = 1$	$k = 5$	$k = 10$	$k = 20$	$k = 50$	$k = \infty$
EDC	0.01	0.06	0.1	0.2	0.6	8.6
EDC2	0.03	0.1	0.3	0.6	1.5	—
ADC	0.04	0.2	0.4	0.7	1.8	24.7
ADC2	0.04	0.2	0.4	0.7	1.8	—
HDC	0.02	0.09	0.2	0.4	0.9	8.9
HDC2	0.02	0.1	0.2	0.5	1.5	—

Table 5.3: FPRs obtained for Meridian with different values of k and different criteria (in $\%$)

Since the results presented in section 5.7.2 have shown that our best one-hop shortcut detection criterion is either HDC, or HDC2, we will just compare these two criteria here. On P2PSim, we see that HDC applied with $k = 20$ and HDC2 applied with $k = 10$ provide both a FPR equal to 0.4%. Thus, the number of useless measurements performed by these criteria are identical and we have a tie between them on P2PSim. On Meridian, we see that HDC applied with $k = 20$ provide a FPR equal to 0.4% while HDC2 applied with $k = 10$ provide a FPR equal to 0.2%. Thus, on the Meridian data set HDC2 is a little bit more efficient than HDC.

The fact that the additional measurements have less impact on the Meridian data set than on the P2PSim data set is normal. Indeed, in Meridian, there exists a huge amount of shortcuts (about 25% of the triangles are TIVs) and there exists a shortcut (resp. an interesting shortcut) for 97% (resp. 83%) of the paths. Thus, considering systematically k candidates generates only a small amount of false positives. On P2PSim, this is a little bit different: there exists less shortcuts (about 12% of the triangles are TIVs) and there exists a shortcut (resp. an interesting shortcut) for 86% (resp. 43%) of the paths. Obviously, considering systematically k candidates generates more false positives on P2PSim than on Meridian.

[6]Note that computing the FPR for $k = \infty$ is useless with EDC2, ADC2 or HDC2. Indeed, with these criteria, the set $\mathcal{S}$ contains all the existing C nodes and using $k = \infty$ means considering all the nodes as positives. Obviously that gives a TPR = 100% but also a FPR = 100%.

In conclusion, choosing which is the best criterion between HDC and HDC2 is difficult. Both are good criteria. In the sequel of the thesis we will continue to use HDC applied with $k = 20$ as reference. However, we keep in mind that HDC2 can provide similar detection results and can be applied even in situations where measuring $RTT(A, B)$ is problematic (*e.g.* to optimize all the routes in the network or to do an anticipative search).

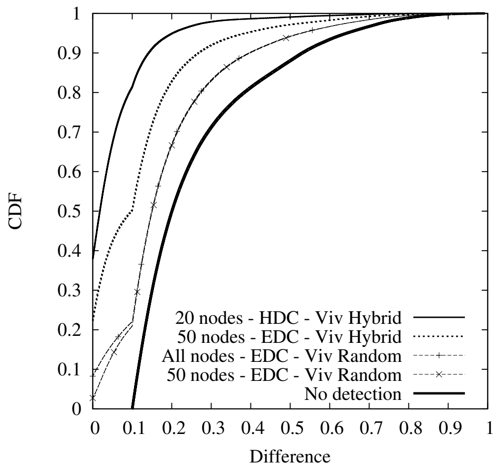
5.8 Reducing the constraint on the neighbors

As discussed in chapter 3, all the previous results have been obtained with a version of Vivaldi using a hybrid neighbors selection scheme: for each node, half of the neighbors are the nearest nodes and the other half of the neighbors are randomly selected nodes. Even if that selection scheme is the one providing the best estimations, it is quite difficult to obtain in practice. Particularly, it is impossible to have such neighbors at the beginning for a node taking part in a Vivaldi system: since it has no measurement available and no coordinate, a node can only start with randomly selected neighbors and it converges progressively towards hybrid neighbors. A node A can obtain such convergence by applying the following procedure each time it has at least one measurement result with each of its m current neighbors:

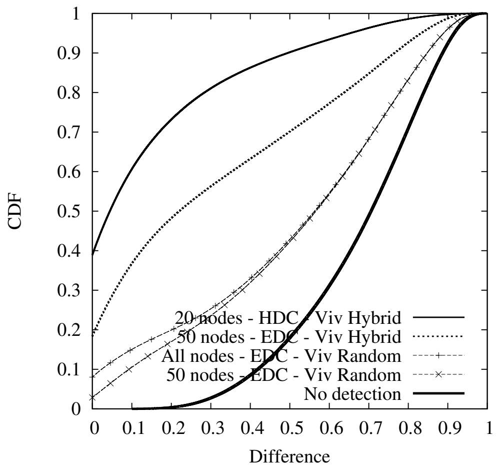
1. Sort A's neighbors by increasing order of measured RTT
2. Keep the $m/2$ first neighbors of the ranking in a set $\mathcal{N}_1$ and discard the others
3. Choose randomly $m/2$ new neighbors and put them in a set $\mathcal{N}_2$
4. $\mathcal{N} = \mathcal{N}_1 \cup \mathcal{N}_2$ is the new set of m neighbors used by A to update its coordinate

After that neighbors update process, A continues to update its Vivaldi coordinate using the new set of neighbors $\mathcal{N}$. When it will have done at least one measurement with each neighbor in $\mathcal{N}$, A will rerun the procedure in order to update its neighbors set and the cycle restarts. Such neighbors update process does not require more measurements than the measurements done by Vivaldi and, from cycles to cycles, $\mathcal{N}_1$ will progressively converge toward the set of A's $m/2$ nearest nodes. But, that convergence takes some time and, at the beginning, a node has no other choice than computing its coordinate with randomly selected neighbors. In this section, we will evaluate if running Vivaldi with random neighbors instead of running it with hybrid neighbors has an impact on the detection results.

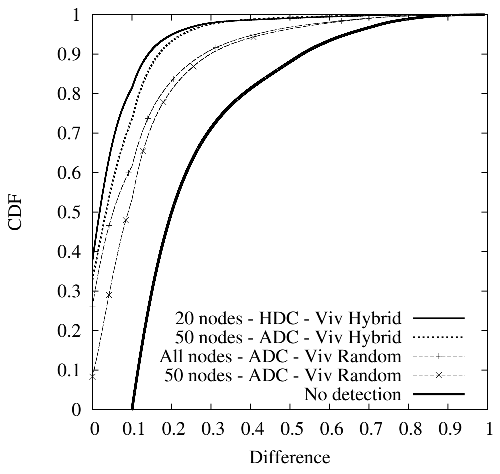
To evaluate this, we will compute the potential improvement remaining after the detection process. The only difference with the empirical study presented in section 5.3.2 is the way the coordinates are computed: in section 5.3.2, we used Vivaldi with hybrid neighbors and, here, we will use Vivaldi with randomly selected neighbors. As usually with a random choice, we repeated the study with different random inputs. Since the results we obtained with these different random neighbors selections were very similar, we present only one of them here. For the P2PSim data set (resp. the Meridian data set), figure 5.7(a) (resp. figure 5.7(b)) shows the results obtained with EDC, figure 5.7(c) (resp. figure 5.7(d)) shows the results obtained with ADC and figure 5.7(e) (resp. figure 5.7(f)) shows the results obtained with HDC.



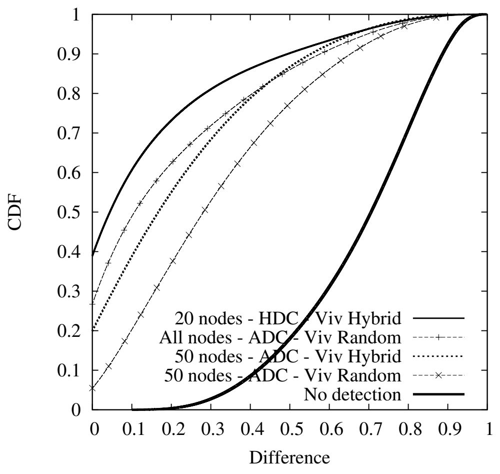
(a) EDC - P2PSim



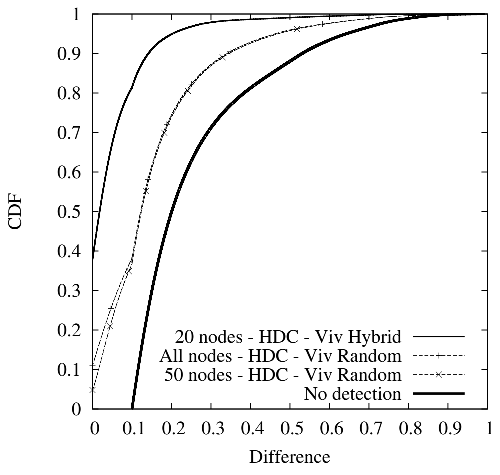
(b) EDC - Meridian



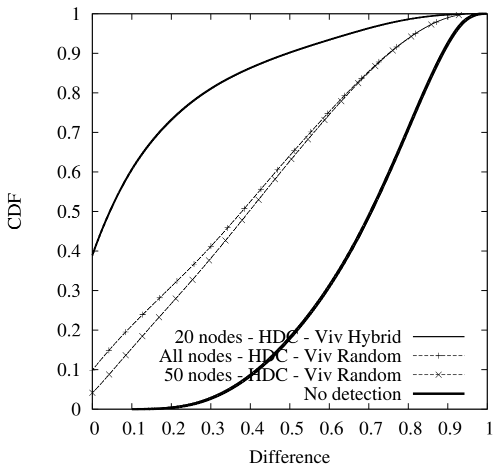
(c) ADC - P2PSim



(d) ADC - Meridian



(e) HDC - P2PSim



(f) HDC - Meridian

Figure 5.7: Difference of G_r between the best existing shortcut and the best detected shortcut when the criteria are applied with estimations produced by a Vivaldi where neighbors are randomly selected.

The main observation in figure 5.7 is that, with every criterion, the detection results obtained with Vivaldi applied using randomly selected neighbors are significantly worse that the detection results obtained with Vivaldi applied using hybrid neighbors. For example, with EDC on P2PSim (figure 5.7(a)), even if we consider all the C nodes in $\mathcal{S}$ as potential shortcuts (*i.e.* $k = \infty$) when Vivaldi is applied with randomly selected neighbors, the obtained results are worse than those obtained by considering $k = 50$ when Vivaldi is applied with hybrid neighbors: with hybrid neighbors and $k = 50$, we have a potential improvement remaining after the detection process which is smaller than 20% of $RTT(A, B)$ for about 85% of the paths AB while, with random neighbors and $k = \infty$, we have a potential improvement remaining after the detection process which is smaller than 20% of $RTT(A, B)$ for only about 70% of the paths AB.

In conclusion, working with Vivaldi using randomly selected neighbors is not a good idea. Since we do not have the choice, we will deal with it at the beginning of a coordinate computation but converging towards hybrid neighbors is mandatory to ensure reliable shortcut detection results.

5.9 Suitable estimations for our criteria

In the previous section, we have compared two estimation mechanisms: on one side we had Vivaldi applied with hybrid neighbors and, on the other side Vivaldi applied with randomly selected neighbors. Even if it was Vivaldi on both sides, the estimations produced with different neighbors selection schemes have very different characteristics. We saw that it has a significant impact on the quality of the results produced by our detection criteria and, in the sequel, we will investigate the detection results obtained with other estimation mechanisms. To choose an suitable estimation mechanism, we have to know which characteristics the estimations must have in order to generate good detection results with our criteria. This is what we will investigate now.

For a given path AB, with each criterion, estimations are only used to compute an approximation for every alternative paths ACB. The way the estimations are used depends of the criterion. Consequently, good estimations for a given criterion will not necessarily be good for the other criteria. Nevertheless, the estimations must always allow us to detect the best shortcuts. Intuitively, a good shortcut for a path AB is a node C which is simultaneously near A and near B. In other words, AC and CB are short edges. Thus, in the following discussion, we will focus on the approximations obtained for the short edges using the estimations.

5.9.1 Suitable estimations for EDC

With EDC, the RTT of an alternative path ACB is approximated by $EST(A, C) + EST(C, B)$. Consider that C is a shortcut for the path AB. If the alternative path ACB is under-estimated by the estimation mechanism, there is no problem: since

$$RTT(A, B) > RTT(A, C) + RTT(C, B) > EST(A, C) + EST(C, B)$$

C will be detected as a potential shortcut when EDC verifies inequation (5.2). If the alternative path ACB is only a little bit over-estimated, then

$$RTT(A,B) > EST(A,C) + EST(C,B) > RTT(A,C) + RTT(C,B)$$

and C is still detected as a potential shortcut by EDC when it verifies inequation (5.2). On the other hand, if the alternative path ACB is significantly over-estimated, then

$$EST(A,C) + EST(C,B) > RTT(A,B) > RTT(A,C) + RTT(C,B)$$

and C will not be detected as a potential shortcut by EDC. Defining exactly which level of over-estimation becomes problematic is difficult because it depends of the gain provided by the shortcut: a shortcut providing an important gain will admit a larger over-estimation before inequation (5.2) rejects it. However, even a shortcut providing an important gain can still be considered as a shortcut if it is significantly over-estimated, its estimated gain will appear small in such situation. Thus, when we will rank the potential shortcuts with respect to the estimated gain, this shortcut will probably not appear in the first positions of the ranking and it will finally be discarded when we will consider only the first k candidates of the ranking.

In conclusion, over-estimations of the alternative paths ACB where C is a shortcut for AB are always problematic for EDC (either during the verification of inequation (5.2), either during the ranking of the candidates). Since intuitively the best shortcuts for a path AB are such that AC and CB are short edges, we conclude that a suitable estimation mechanism for EDC, is *an estimation mechanism that minimizes the over-estimation of the small paths*.

5.9.2 Suitable estimations for ADC

With ADC, the RTT of an alternative path ACB is approximated by $RTT(A, C_A) + RTT(B, C_B)$ where C_A (resp. C_B) is C's nearest node among A's (resp. B's) Vivaldi neighbors according to the estimated RTTs.

More than the quality of the estimations, the crucial point with ADC is the neighbors selection scheme. In order to obtain a reliable approximation for ACB's RTT there must exist at least one of A's (resp. B's) neighbor that is near C. Since the best shortcuts C for a path AB are intuitively such that AC and CB are short edges, we can conclude that hybrid neighbors will provide best results with ADC than randomly selected neighbors. Indeed, with randomly selected neighbors, there is a large probability that all A's (resp. B's) neighbors are far from the node. Thus, when A (resp. B) will have to choose a node C_A (resp. C_B) it will have no other choice than taking a node which is far from it. Since the C nodes providing the best shortcuts are intuitively near A (resp. B), approximating $RTT(A, C)$ (resp. $RTT(C, B)$) by $RTT(A, C_A)$ (resp. $RTT(B, C_B)$) in such conditions will lead to large approximations for the paths ACB. Thus, the problem will be the same as the problem experienced by EDC with over-estimations of the short edges: either the best shortcuts will not be detected as candidates, or they will not appear in the first positions of the ranking of the candidates. In both cases, the best shortcuts will not be detected by ADC if the estimation mechanism works with randomly selected neighbors.

Even though they are not the most important point for ADC, estimation errors can also impact the detection results. Indeed, estimated RTTs are used to select C_A (resp. C_B) among A's (resp. B's) neighbors. If there are estimation errors, then C_A (resp. C_B) may be a node which is not C's nearest node among A's (resp. B's) neighbors. This leads to a bad approximation of ACB's RTT and to bad detection results. For a given path AB and a given node C, there are 32 possibilities for C_A and 32 possibilities for C_B. For C_A (resp. C_B), among these 32 possibilities, ADC has to find C's nearest node. In other words, ADC has to find the smallest path among 32 C_AC (resp. C_BC) paths. To do that, ADC ranks the paths by increasing order of estimated RTT and take the first path of the ranking. If the path which has the smaller RTT has also the smaller estimated RTT, then ADC chooses the good C_A (resp. C_B). Consequently, like EDC, ADC requires an estimation mechanism that does not tend to over-estimate the small paths.

In conclusion, ADC requires two things to be able to detect the best shortcuts. First, *a hybrid neighbors selection scheme* in order to have suitable C_A and C_B available for the C nodes that are shortcuts. Secondly, *an estimation mechanism that minimizes the over-estimation of the small paths* in order to be able to choose the most suitable C_A and C_B for any given C.

5.9.3 Suitable estimations for HDC

HDC's principle is, by default, to approximate AC and CB as ADC does. But, if it is impossible to find a C_A (resp C_B) sufficiently near C, then HDC approximates AC (resp. CB) as EDC does. Since HDC behaves by default like ADC, HDC's requirements about estimations are intuitively the same as ADC's requirements. So, the most crucial point with HDC is to work with hybrid neighbors. If this condition is satisfied, having an estimation mechanism that minimizes the over-estimation of the small paths may improve the detection results.

However, if the estimation mechanism works with randomly selected neighbors, HDC can provide better results than ADC. Indeed, in such situation, we know that ADC generally fails to find suitable C_A (resp. C_B) for the C nodes that are the best shortcuts for the path AB. It leads to an over-estimation of the alternative paths providing the best shortcuts and it gives bad detection results. In such situation, HDC avoids ADC's large approximation errors because it switches to the EDC behaviour. So, if we have an estimation mechanism working with randomly selected neighbors but which does not tend to over-estimate the small paths, HDC can still provide good detection results. Nevertheless, since HDC will behave like EDC when it has to decide if the best shortcuts are shortcuts or not, it will give results similar to the one provided by EDC. In other, words HDC is quite useless if we do not use a hybrid neighbors selection scheme and we can directly apply EDC. In conclusion, to obtain good detection results, HDC's requirements about estimations are the same as ADC's requirements.

5.9.4 Impact of the neighbors choices on Vivaldi estimations

In the previous subsections, we have discussed the requirements of our criteria in order to allow them to detect the best shortcuts. In summary, every criteria require as less as possible over-estimations of the small paths and, for ADC and HDC, working with hybrid neighbors seems mandatory. In section 5.8, we have already compared the results obtained with two estimation mechanisms: on one side, Vivaldi working with hybrid neighbors and, on the other side, Vivaldi working with randomly selected neighbors. Figure 5.7 page 97 shows that the detection results obtained with Vivaldi using randomly selected neighbors are systematically worse than those obtained with Vivaldi using hybrid neighbors. These results confirm the fact that ADC and HDC require hybrid neighbors in order to be able to provide accurate detection results.

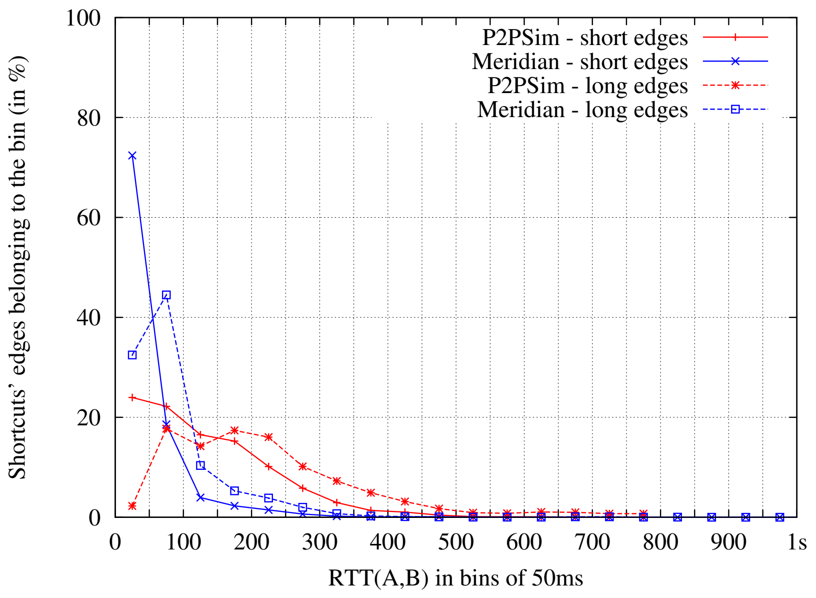
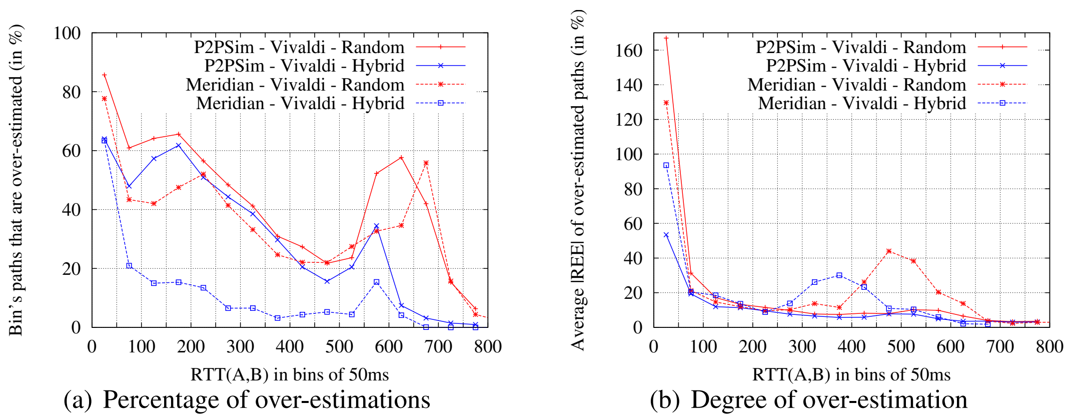


Figure 5.8: Characteristics of the bins for P2PSim and Meridian: the figure shows the distribution of the shortcuts' edges among the bins.

Since EDC does not worry about the neighbors selection scheme, we must analyse the estimation errors. For this analysis, we divide the whole range of RTTs in the P2PSim (resp. Meridian) data set into equal bins of 50 ms each. Figure 5.8 shows the distribution of the shortcuts' edges among the bins for the P2PSim and the Meridian data sets. With Meridian, we see that about 75% of the shortcuts' short edges belong to the first bin. For this data set, it is clear that focussing on the estimation errors for the paths of the first bin will cover a large part of the shortcuts' short edges existing in the network. For P2PSim, it is less clear because the first bin contains only 23% of the total number of shortcuts' short edges. To cover more shortcuts' short edges, we can extend our observations to the first 3 bins. Thus, we cover more than 60% of the total number of shortcuts' short edges.

For each bin, figure 5.9 gives a description of the over-estimations experienced by the bin's paths: figure 5.9(a) shows the percentage of bin's paths that are over-estimated and figure 5.9(b) shows the average of the $|REE|$ observed for bin's paths that are over-



(a) Percentage of over-estimations (b) Degree of over-estimation

Figure 5.9: Over-estimations of the paths obtained with Vivaldi and, either a random, or a hybrid neighbors selection scheme.

estimated. In figure 5.9(a) we see that, on both data sets, a random neighbors selection scheme has a general tendency to over-estimate the small paths. If we focus on the paths of the first bin, we see that 85% (resp. 79%) of the paths are over-estimated in P2PSim (resp. Meridian) when Vivaldi uses such neighbors selection scheme. With a hybrid neighbors selection scheme, the percentage of over-estimated paths is generally smaller. For example, in the first bin, the percentage of over-estimated paths falls around 60% with both data sets when a hybrid neighbors selection scheme is used. So, we know that more small paths are over-estimated when a random neighbors selection scheme is used than when a hybrid neighbors selection scheme is used. But, we do not know anything about the degree of over-estimation: having 80% of the paths that are over-estimated is not a problem if the over-estimation is small on average. Information about the degree of over-estimation is given in figure 5.9(b). In this figure we see that, for the first bin's paths, the average $|REE|$ is big when a hybrid neighbors selection scheme is used: we have means that are about 55% for P2PSim and 95% Meridian. But, the average $|REE|$ is huge when a random neighbors selection scheme is used. With such neighbors, in P2PSim, the average $|REE|$ is bigger than 160% ! Since a small path of 40ms can be estimated at more than 120ms and, since longer paths are less often and less strongly over-estimated, this can be a real problem for EDC: using the estimations to approximate the alternative paths, the smallest ones (*i.e.* the best shortcuts) appears longer than longer paths. Consequently, EDC will be unable to detect the best shortcuts. Anyway, these observations confirm that EDC provides better detection results with an estimation mechanism that tends to minimize the over-estimation of the small paths.

5.10 Conclusion

In this chapter, we have presented three detection criteria based on Vivaldi's estimations that allow one-hop shortcuts detection in network: EDC, ADC and HDC. Since these criteria are based on estimations, they make detection errors and are not able to detect the best existing shortcut for any given path AB. A measurement-based detection mechanism like RON guarantees to find the best existing shortcut for any given path AB, but our approach is a lot more scalable and allows us to obtain a significant RTT improvement for lots of paths. Compared to a simple random relay node selection, we have seen in section 5.6 that our approach can provide much better RTT improvements. In conclusion, we think that our estimation based approach is a good trade-off between scalability and quality of the detection.

In the second part of the chapter, we also studied variants of our detection mechanism. In section 5.7, we have seen that it is possible to obtain similar detection results without measuring $RTT(A, B)$. In section 5.8 we have seen that using estimations provided by Vivaldi working with randomly selected neighbors does not provide as good detection results as using the estimations provided by Vivaldi working with hybrid neighbors. Finally, in section 5.9, we discussed the estimations' required characteristics in order to obtain good estimation results with our criteria: every criterion requires as few as possible over-estimations of the small paths and, for ADC and HDC, working with hybrid neighbors seems mandatory.

In the light of the results, other estimation mechanisms are perhaps more suitable than a vanilla Vivaldi for our criteria. Since shortcuts are not embeddable in a metric space, we should replace Vivaldi by an estimation mechanism that can deal with TIVs (*i.e.* shortcuts). In the following chapters, we will investigate two ways to do that. A first possibility has been proposed by Wang *et al.* [82]. Their idea is to apply non-linear transformations to the delays before trying to embed them in a metric using Vivaldi. Since non-linear transformations eliminate TIVs, a situation that was not embeddable initially can become embeddable after the transformation. We will investigate that in chapter 6. A second possibility is to use an estimation mechanism that has been developped to deal with TIVs. In chapter 4, we have briefly presented one of these estimation mechanisms, namely DMF [42, 40]. DMF sees the estimation problem as a matrix completion problem: measurements between the nodes and their "neighbors" give a few values in the delay matrix and DMF tries to infer the other values of the matrix from the measured values. This way to proceed allows potentially DMF to produce an estimated delay matrix which is an exact reproduction of the measured delay matrix. We will investigate that in chapter 7. Finally, in chapter 8, we will present our hierarchical approach of Vivaldi [7, 29, 6]. That structured approach of Vivaldi provides more accurate estimations for the small RTTs and, in theory, that will generate better shortcuts detection results using our criteria.

Chapter 6

Non-linear transformations

Abstract

Wang et al. [82] presented an interesting idea to allow Vivaldi to deal with TIVs. Their idea consists in applying a non-linear transformation to the delays before trying to embed them in a metric space. Since non-linear transformations allow us to eliminate TIVs, a situation that was initially not embeddable can become embeddable after the transformation. If Vivaldi is able to accurately embed the transformed delays, then we should restore the TIVs by applying the reverse transformation to the obtained estimations. If that would work, we would not need any one-hop shortcut detection criteria any longer: we would simply use the estimated delays to search the shortcuts exactly as we would do with measured delays. In this chapter, we will investigate that idea. We will analyse the quality of the estimations it generates and we will see if that idea can improve (or not) the shortcut detection results we obtained with HDC using the estimations produced by a vanilla Vivaldi. Most of the results presented in this chapter have been published in [9].

6.1 Improving the accuracy of an ICS using non-linear transformations

6.1.1 Principle

The idea proposed in [82] consists in eliminating TIVs by applying non-linear transformations to the delays. This idea is based on the fact that a non-linear transformation can stronger decrease long delays than short delays. In other words, for a given TIV ACB, it can strongly reduce the RTT of the long edge (AB), while it applies a smaller reduction to the RTTs of the small edges (AC and CB) Thus, as indicated in figure 6.1, non-linear transformations allow us to eliminate TIVs.

For a given delay matrix, this approach allows potentially to obtain a transformed matrix that contains no TIVs (or, at least, less TIVs than the initial matrix) and from which it is possible to restore the initial matrix by applying the reverse transformation. In the light of this observation, there could be a simple solution to improve the accuracy of

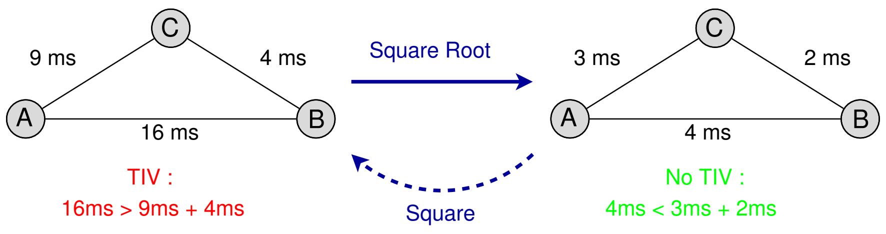


Figure 6.1: **Example of TIV elimination using a non-linear transformation.** In this figure we can see how a simple non-linear transformation (in this case a square root) can transform a non-embeddable situation in an embeddable situation.

the estimations obtained with a classical ICS. We saw in chapter 3 that the principle of an ICS is to take a few measurements in a delay matrix, to try to embed these measurements in a metric space and to infer the RTT of any given path from this embedding. In other words, from a few elements taken in a *Measured Delays matrix* (MD matrix), it builds a complete *Estimated Delays matrix* (ED matrix). But, rather than embedding the delays of the MD matrix, the ICS can embed the delays of the *Transformed Measured Delays matrix* (TMD matrix). Since the TMD matrix contains no TIVs (or only a small number of TIVs compared to the MD matrix), the embedding should be easier for the ICS. Thus, the *Transformed Estimated Delays matrix* (TED matrix) infered from the embedding of the TMD matrix should provide an accurate estimation of the TMD matrix. Finally, if we apply the reverse transformation to the TED matrix estimations, then we should obtain an ED matrix which provides accurate estimations of MD matrix delays (including the TIVs). The non-linear transformation approach is summarized in figure 6.2.

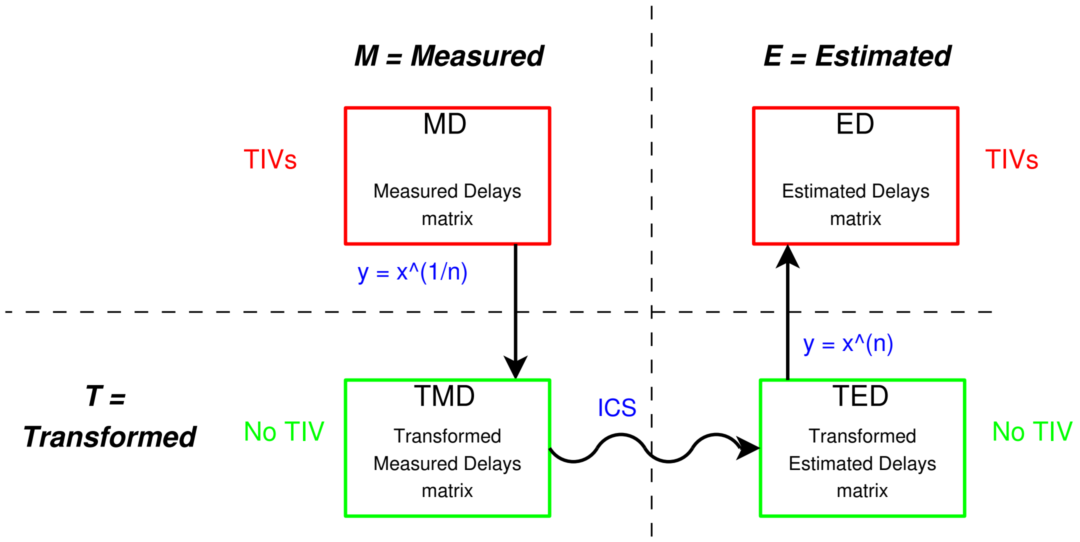


Figure 6.2: **Using a non-linear transformation to improve the accuracy of an ICS.** In this figure we can see the steps of a non-linear transformation-based approach for an ICS. If the ICS is able to build a TED matrix which is an exact reproduction of the TMD matrix, then the ED matrix is an exact reproduction of the MD matrix and we have an estimation mechanism that can deal with TIVs.

6.1.2 Eliminating TIVs using non-linear transformations

In figure 6.1, we have shown that it is possible to eliminate TIVs with a simple non-linear transformation like a square root. However, in this example, the gain provided by the TIV was small. To eliminate TIVs providing a more important gain (*i.e.* more interesting shortcuts), it may be necessary to apply a non-linear transformation which is stronger than a square root (see the example in figure 6.3). For this study, we will consider only $y = x^{1/n}$ transformation functions (for different values of parameter n). Such functions have many advantages. First, it allows us to eliminate lots of TIVs because big values are much more reduced than small values by such function. Secondly, it can be easily reversed to compute the ED matrix from the TED matrix. Finally, since it is a monotonically increasing function, we have the guarantee that it will not generate new TIVs in the TMD matrix. Indeed, for any triangle ACB, the longer edge AB is still the longer edge after the transformation and, since big values are more decreased than the small values during the transformation, it cannot generate new TIVs.

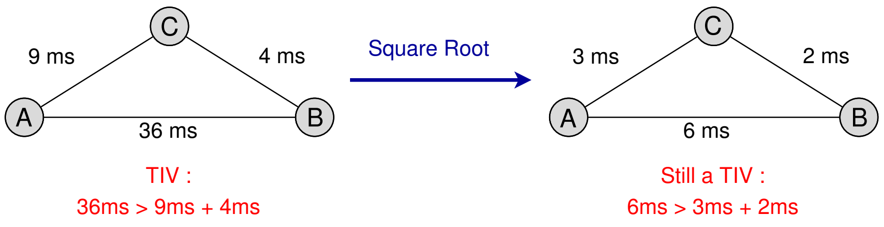
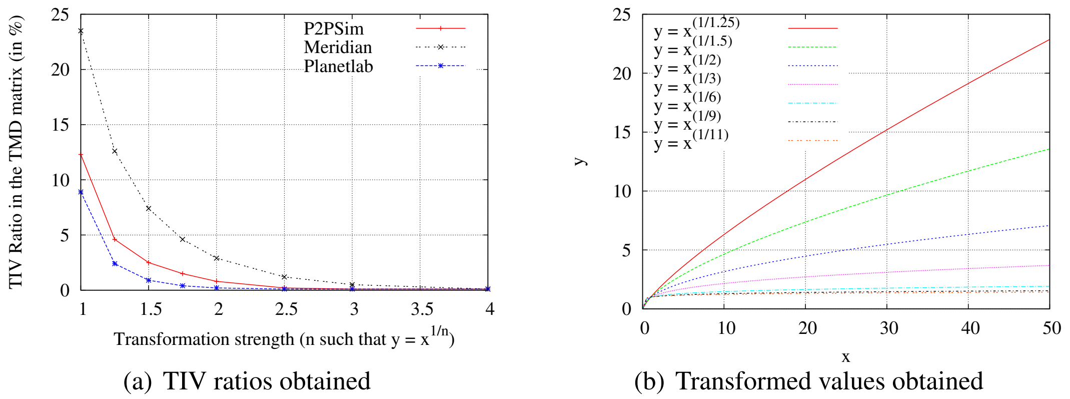


Figure 6.3: **Example where a square root is not sufficient to eliminate a TIV.** In this figure we can see that stronger non-linear transformations than a square root are sometimes necessary to eliminate TIVs.

In [82], Wang *et al.* illustrate their idea using the square root. We will now investigate how such transformation can reduce the TIV ratio (*i.e.* the proportion of triangles that are TIVs) in our matrices. In chapter 4, we have seen that the TIV ratio in our delay matrices is respectively 12.3% for P2PSim, 23.5% for Meridian and 8.9% for Planetlab. Non-linear transformations should reduce these values. Figure 6.4(a) shows, for each matrix, the TIV ratio of the TMD matrices obtained with different $y = x^{1/n}$ transformation functions. In this figure, we see that a square root eliminates lots of TIVs in our matrices but not all of them: the TIV ratio in the TMD matrix obtained with a square root is 0.8% for P2PSim, 2.9% for Meridian and 0.2% for Planetlab. It is possible to eliminate more TIVs with non-linear transformations stronger than a square root. The smallest value of n so that the transformation $y = x^{1/n}$ eliminates all the TIVs of the matrix is $n = 9$ for P2PSim, $n = 11$ for Meridian and $n = 6$ for Planetlab. For each matrix, we call the transformation corresponding to that minimal value of n the *"no TIV" transformation*. But, the "no TIV" transformation is really extreme and, in figure 6.4(a), we see that a cube root is generally sufficient to eliminate most of the TIVs in every data set. With a cube root, the TIV ratio in the TMD matrix is 0.1% for P2PSim, 0.5% for Meridian and less than 0.1% for Planetlab.

Another important characteristic of a transformation function is the transformed values it generates. Indeed, in figure 6.4(b), we see that the curves corresponding to the



(a) TIV ratios obtained (b) Transformed values obtained

Figure 6.4: **Impacts of non linear transformations.** Figure 6.4(a) shows the TIV ratios in the TMD matrices obtained with different transformation functions. Figure 6.4(b) shows the types of values obtained in the TMD matrices with different transformation functions.

stronger transformation functions have a very low slopes. Thus, even if these transformations allow us to eliminate all the TIVs, the delays in the obtained TMD matrix are values that are very near each other: at the limit, with a very strong transformation, all the values in the TMD matrix are approximately equal to 1. This has two major consequences. Firstly, even if the TMD matrix contains no TIVs (or only a few TIVs), the ICS will have difficulties to embed its delays in a given metric space. Indeed, in example 4 page 50, we have seen that a $N-1$ dimensional Euclidean space is necessary to perfectly embed a network composed of N equidistant nodes. Working with such embedding space is not realistic for large networks. Consequently, we will obtain potentially large estimation errors even if the TMD matrix contains no TIV. The second consequence is that small estimation errors in the TED matrix can generate very large estimation errors in the ED matrix. Let's consider that we use $y = x^{1/n}$ as transformation function. To compute the ED matrix from the TED matrix, we must apply the transformation $x = y^n$. But, in the TED matrix, there are estimation errors. Let ϵ be the relative estimation error experienced by a given delay y. In other words, if y is the measured delay in the TMD matrix, then its estimation in the TED matrix is equal to $y(1+\epsilon)$. During the transformation applied to obtain the ED matrix, this error becomes:

$$(y(1+\epsilon))^n = y^n(1+\epsilon)^n \approx y^n(1+n\epsilon) \quad \textit{for } \epsilon \lll 1$$

Thus, at first approximation, the relative errors in the TED matrix are multiplied by n in the ED matrix. Since our first observation was that the embedding of the TMD matrix will not be as easy as it seems to be, we can finally obtain very large estimation errors in the ED matrix.

6.2 Quality of the estimations

In this section, we will investigate if non-linear transformations allow to improve the accuracy of the estimations obtained with Vivaldi. For this study, we will consider a few non-linear transformations. Since it is the transformation used in [82], we will consider the square root ($n = 2$) as a reference. From the analysis of figure 6.4(a), we decided to consider the "no TIV" transformation ($n = 9$ for P2PSim, $n = 11$ for Meridian and $n = 6$ for Planetlab) and the cube root ($n = 3$) because it eliminates most of the TIVs in every data set. We will also analyse the results obtained with two transformations that are weaker than the square root: we will consider the TMD matrices obtained with $n = 1.5$ and $n = 1.25$.

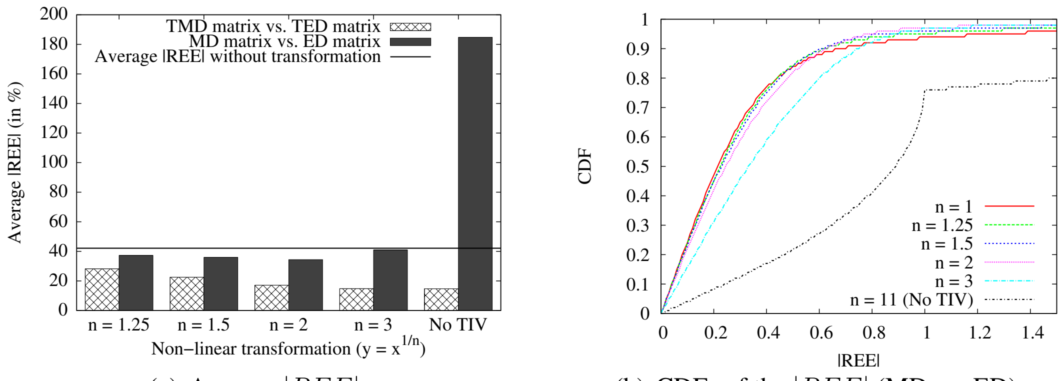
Thus, for each MD matrix (*i.e.* P2PSim, Meridian and Planetlab), we build five TMD matrices (one for each transformation function considered). Then, we try to estimate these matrices using Vivaldi and we obtain five TED matrices for each data set. From the TED matrices, we obtain ED matrices by applying the suitable reverse transformation. For each MD matrix and each transformation function, we will compute two estimation errors: the $|REE|$ between the TMD matrix and the TED matrix and the $|REE|$ between the MD matrix and the ED matrix. In the following subsections, for each comparison, we will give the average $|REE|$ obtained for the paths and the CDF of the $|REE|$ of the paths.

6.2.1 Vivaldi using a Euclidean space

For our first tests, we tried to embed the TMD matrices in a 9-dimensional Euclidean space. To compute its coordinate, each node uses 32 neighbors with a hybrid neighbors selection scheme. Figure 6.5(b) gives the CDFs of the $|REE|$ obtained for Meridian paths when we compare ED to MD matrix delays (similar results have been obtained for the other data sets but the curves are not presented here). Figure 6.5(a) (resp. 6.6(a)) gives the average $|REE|$ obtained for Meridian paths (resp. Planetlab paths) with different non-linear transformations (similar results have been obtained for P2PSim but the figure is not presented here). For each transformation, the white box gives the average $|REE|$ obtained when the TED matrix is compared to the TMD matrix and the black box gives the average $|REE|$ obtained when the ED matrix is compared to the MD matrix. The horizontal line represents the average $|REE|$ obtained without applying any transformation to the MD matrix (*i.e.* the results obtained with a vanilla Vivaldi).

Let us begin with the observation of the estimation errors between the TED and the TMD matrices (white boxes on figures 6.5(a) and 6.6(a)). A general trend is that Vivaldi makes smaller estimation errors when it works with the transformed matrices than when it works with the original matrix (without applying any transformation). For P2PSim and Planetlab, the estimations in the TED matrix are more accurate with weak transformations (*i.e.* small values of n) than with strong transformations. As discussed in section 6.1.2, the strongest transformations produce TMD matrices where all the values are contained in a small interval and such TMD matrices are more difficult to embed.

Since the estimation errors observed in the TED matrices are quite small, we hope that the ED matrices obtained using non-linear transformations provide better estimations than the ED matrices obtained with a vanilla Vivaldi. According to figures 6.5(a) and 6.6(a), this is effectively the case: for the weaker transformations, the black boxes indicate values that are smaller than what we can obtain with a vanilla Vivaldi (the horizontal line). But,



(a) Average $|REE|$ (b) CDFs of the $|REE|$ (MD vs. ED)

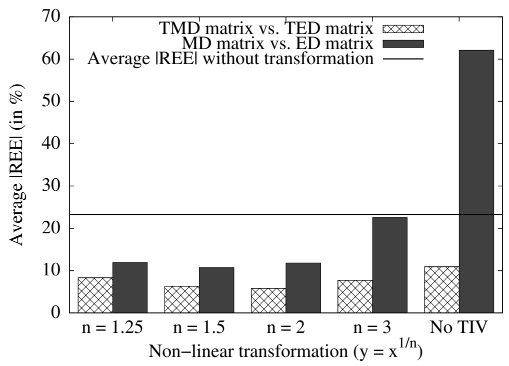
Figure 6.5: **Quality of the estimations obtained for Meridian paths using a 9D Euclidean space.** Figure 6.5(a) gives the average $|REE|$ obtained for Meridian paths before and after transformations. Figure 6.5(b) compares the estimations of the ED matrices to the delays of the MD matrix and gives the CDFs of the obtained $|REE|$.

this is not with the strongest transformations that we obtain the best results. With P2PSim, we obtain the best estimations with $n = 1.25$: the average $|REE|$ is 1.2% smaller than with a vanilla Vivaldi. For Meridian, according to figure 6.5(a), the best results seem to be obtained with a square root. However, if we analyse the CDFs in figure 6.5(b), we see that the square root CDF is worse than the CDF obtained without applying any transformation for about 85% of the paths. In the light of this observation, we consider that the best choice for Meridian is to use a non-linear transformation with $n = 1.5$. For Planetlab, according to figure 6.6(a), using $n = 1.5$ seems the best choice: the average $|REE|$ is reduced of 12.6% compared to what we obtain with a vanilla Vivaldi.

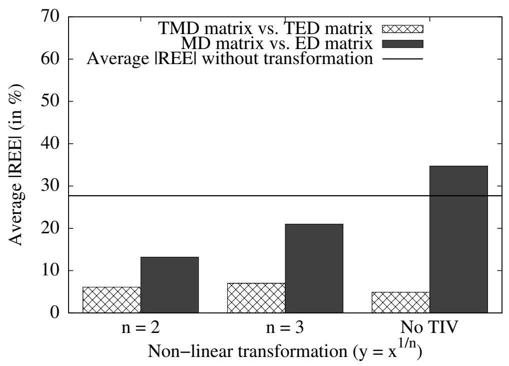
In summary, we have observed that non-linear transformations allow to reduce the average $|REE|$ compared to what we obtain with a vanilla Vivaldi. For P2PSim, we have only a very small improvement (1.2%) while we have a large improvement with Planetlab (12.6%). For Meridian, the improvement is significant but the average $|REE|$ obtained using non-linear transformations is still important: with $n = 1.5$, we have an improvement of 6% but the average $|REE|$ is still equal to 36%.

6.2.2 Vivaldi using a space augmented with a height

In section 6.2.1, we have seen that the weakest transformations provide the best results in a Euclidean space. However, since a $N-1$ Euclidean dimensional space is required to allow a perfect embedding for a topology composed of N equidistant nodes, strong transformations cannot provide good results with a Euclidean embedding space. Other types of spaces may be more suitable to embed matrices where all the delays are approximately the same. Dabek *et al.* [15] have proposed a type of space that seems suitable for this purpose: a Euclidean space augmented with a height. We have described this space in section 3.3.3. In such space, each node has a coordinate $[\vec{x}, x_h]$ where $\vec{x}$ is the coordinate of the node x in a Euclidean space and x_h is the height of the node x. A packet sent from



(a) 9D Euclidean space



(b) 8D Euclidean space augmented with a height

Figure 6.6: **Quality of the estimations obtained for Planetlab paths.** Figure 6.6(a) (resp. 6.6(b)) gives the average $|REE|$ obtained for Planetlab paths before and after transformations when a 9D Euclidean space (resp. a 8D Euclidean space augmented with a height) is used.

one node to another must travel the source node's height, then travel in the Euclidean space and, finally, travel the destination node's height. Consequently, it is easy to embed a topology of N equidistant nodes in such type of space: if there is a delay d between each pair of nodes, we have just to give the same Euclidean coordinate to each node and to give them a height equal to $d/2$. This is not so simple in practice[1], but, anyway, it should be possible to obtain smaller estimation errors with the strongest transformations if we use a space augmented with a height rather than a simple Euclidean space.

For the three strongest non-linear transformations for each data set (square root, cube root and "No TIV"), we will try to use a Vivaldi working with a 8D+h embedding space to compute the TED matrix. To compute its coordinate, each node still uses 32 neighbors with a hybrid neighbors selection scheme. Figure 6.6(b) gives the average $|REE|$ of the ED matrices obtained with such embedding space for the Planetlab data set. Similar results have been obtained for our other data sets but the figures are not presented here. The observations and the conclusions for P2PSim and Meridian are the same as those obtained for Planetlab.

Our main observation is that the estimations in the TED matrices are generally more accurate when a 8D+h embedding space is used than when a 9D Euclidean space is used. Nevertheless, the accuracy improvement is not sufficient to compensate the factor n applied to the error when the ED matrix is computed from the TED matrix. Moreover, even if the results obtained with strong transformations and a 8D+h space are better than those obtained with strong transformations and a 9D space, they are still worse than those obtained with weak transformations and a 9D space. Thus, in the sequel of the chapter, we will work with a classical Euclidean space.

[1]Indeed, we know that the height model has constraints in order to perform correctly. Among these constraints, there is one which states that a node cannot have exactly the same Euclidean coordinate as one of its neighbors. So, our simple solution to embed a topology of equidistant nodes will not perform as well as we hope.

6.3 Shortcut detection

In section 6.2, we have seen that non-linear transformations may improve the accuracy of the estimations obtained with an ICS but, unlike what is stated in [82], this does not seem to be the magical solution that allows Vivaldi to deal with TIVs. Anyway, basically, non-linear transformations are a principle that allows us to eliminate TIVs before the embedding and to generate an ED matrix that contains TIVs. In section 6.3.1, we will see if the TIVs in the ED matrix are the same as those existing in the MD matrix, *i.e.* we will see if searching one-hop routing shortcuts among ED matrix's estimations can provide better results than searching these shortcuts using our detection criteria.

6.3.1 Searching shortcuts among the estimated delays

To obtain an ED matrix which contains exactly the same TIVs as the MD matrix, it is necessary to have a TMD matrix which contains no TIV and the TED matrix must contain no estimation error. Following the results presented in section 6.2, we are far from that. However, Wang *et al.* state in [82] that using the square root as tranformation function allows them to generate an ED matrix with a TIV ratio similar to the MD TIV ratio. Indeed, with Meridian (the matrix used in [82]), the TIV ratio in the ED matrix obtained using a square root is equal to 28.3%. This is quite similar to the MD TIV ratio (*i.e.* 23.5%). Nevertheless, the observation is not the same with P2PSim and Planetlab. With these data sets, using a square root as transformation function generates ED matrices where the TIV ratio is respectively equal to 29.6% (rather than 12.3%) and 25.2% (rather than 8.9%). Since the TIVs that are not eliminated during the transformation can, in theory, not be restored in the ED matrix and, since the TIVs eliminated by the transformation can only be restored if the TED contains no estimation errors, we conclude that the good results presented in [82] for the Meridian data set cannot be generalized. Moreover, the TIV ratio only indicates that the percentage of triangles that are TIVs in the ED matrix is similar to the percentage of triangles that are TIVs in the MD matrix. Nothing indicates that they are the same TIVs. To check if the TIVs in the ED matrix are the same as the TIVs in the MD matrix, we introduce a new shortcut detection criterion called SDC (Simple Detection Criterion). With SDC, a node C is considered as a shortcut for the path AB if

$$EST(A,B) > EST(A,C) + EST(C,B) \tag{6.1}$$

It is worth mentionning that this criterion can only detect shortcuts/TIVs if the ED matrix contains TIVs. Applying SDC with the estimations computed with a vanilla Vivaldi (as in chapter 5), returns systematically "there exists no shortcut" for any given path AB. However, the detection results obtained with SDC can be directly compared to the results obtained with the other criteria presented in chapter 5.

For each pair of MD and ED matrices, the columns labelled "TPR" (True Positive Rate) in table 6.1 give the percentage of TIVs ACB existing in the MD matrix that are TIVs in the ED matrix. In other words, it represents the percentage of MD TIVs that can be discovered by searching TIVs in the ED matrix. On the other hand, the columns labelled "FPR" (False Positive Rate) give the percentage of triangles ACB which are not

TIVs in the MD matrix and that are TIVs in the ED matrix. In other words, it represents the percentage of triangles ACB that are wrongly considered as TIVs when we search TIVs in the ED matrix. Obviously, a good detection criterion provides simultaneously a low FPR and a high TPR.

	P2PSim		Meridian		Planetlab	
	TPR	FPR	TPR	FPR	TPR	FPR
$n = 1.25$	10.3%	1.4%	8.7%	1.3%	19.9%	0.8%
$n = 1.5$	21.9%	4.1%	18.6%	3.4%	38.7%	2.7%
$n = 2$	34.1%	8.8%	31.7%	7.5%	49.9%	7.1%
$n = 3$	41.6%	15.8%	40.9%	14.1%	50.5%	15.3%
No TIV	42.9%	28.6%	40.1%	29.5%	48.6%	25.2%

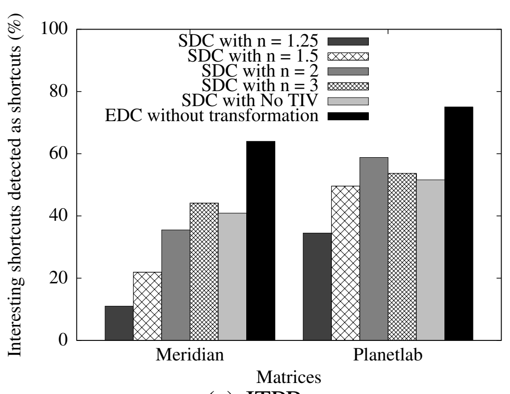
Table 6.1: Detection results using SDC

In table 6.1, we see that simply searching TIVs in the ED matrix allows rarely to detect more than 50% of the TIVs existing in the MD matrix. Moreover, it is difficult to obtain a good TPR without having a significant FPR. These bad detection results show that the ED matrices obtained using non-linear transformations do not contain all the TIVs existing in the corresponding MD matrix and that these ED matrices also contain TIVs which are not existing in the corresponding MD matrix. Finally, we notice that the results obtained with SDC using non-linear transformations are clearly worse than those obtained with the criteria presented in chapter 5 applied using the estimations provided by a vanilla Vivaldi. For example, with EDC on P2PSim, we saw in section 5.3.1 that the TPR is equal to 53% and the FPR is equal to 2%. Following the values presented in table 6.1, there exists no non-linear transformation that provides such results. If we focus on the ITPR (*i.e.* the percentage of interesting shortcuts detected as shortcuts), the conclusion is the same. Indeed, figure 6.7(a), we see that the ITPR obtained by using SDC applied with non-linear transformations is systematically lower than the ITPR obtained with EDC applied with a vanilla Vivaldi.

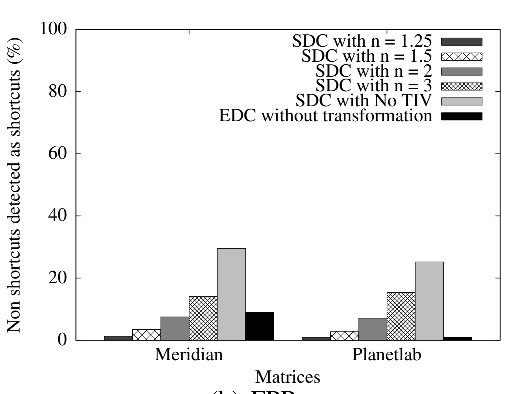
6.3.2 Searching shortcuts using our detection criteria

In section 6.3.1, we have seen that SDC applied with estimations computed using non-linear transformations is not able to provide as good detection results as our criteria presented in chapter 5 applied with estimations computed using a vanilla Vivaldi. But, in section 6.2, we have seen that non-linear transformations are sometimes able to slightly improve the accuracy of the estimations compared to the estimations provided by a vanilla Vivaldi. Consequently, our criteria applied with estimations computed using non-linear transformations could provide better detection results than our criteria applied with estimations computed using a vanilla Vivaldi. We will now investigate that.

Figure 6.8(a) (resp. 6.8(b)) shows the ITPR (resp. FPR) obtained with EDC applied with or without non-linear transformations. We observe that non-linear transformations



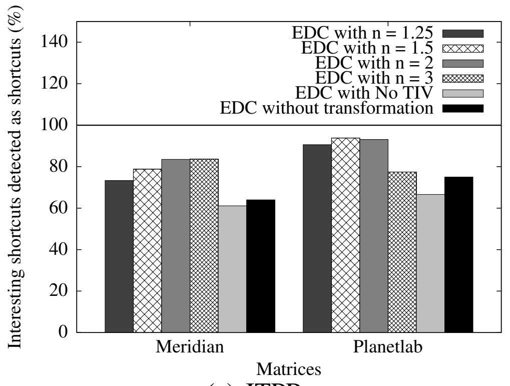
(a) ITPR



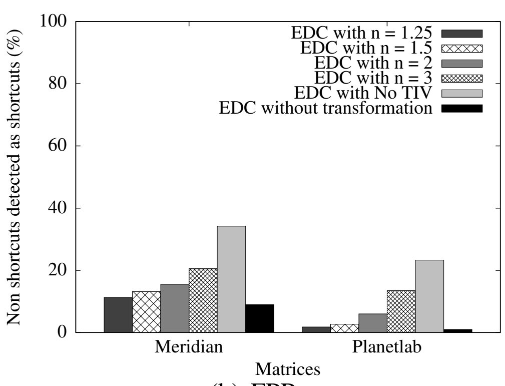
(b) FPR

Figure 6.7: **Comparison between SDC and EDC.** Considering different detection criteria combined with different estimation mechanisms, figure 6.7(a) gives the ITPR and figure 6.7(b) gives the FPR.

improve the detection results. On Meridian, EDC applied with $n = 1.5$ (resp. with a vanilla Vivaldi) generates an ITPR equal to 79% (resp. 64%) and a FPR equal to 13% (resp. 9%). On Planetlab, EDC applied with $n = 1.5$ (resp. with a vanilla Vivaldi) generates an ITPR equal to 94% (resp. 75%) and a FPR equal to 3% (resp. 1%). Thus, on both data sets, EDC applied with non-linear transformations (compared to EDC applied with a vanilla Vivaldi) increases significantly the TPR while it increases only slightly the FPR. With P2PSim, we have not been able to increase the quality of the detection results using non-linear transformations.



(a) ITPR



(b) FPR

Figure 6.8: **EDC applied with non-linear transformations.** Considering EDC combined with different estimation mechanisms, figure 6.8(a) gives the ITPR and figure 6.8(b) gives the FPR.

In conclusion, using non-linear transformations could really improve our detection results in some cases (but not in all cases). However, since non-linear transformations are difficult to parameterize in practice (the optimal value of n depends of the data set) and, since it does not improve the detection results in all data sets, we have chosen to stop our investigations here. Thus, in the sequel, we will still use the detection results obtained with a vanilla Vivaldi as reference.

6.4 Conclusion

In this chapter, we have observed the advantages and the disavantages of the approach proposed in [82] when simple $y = x^{1/n}$ non-linear transformations are applied. On one side, it is clear that applying non-linear transformations to the delays before trying to embed them eliminates TIVs and increases the accuracy of the estimations. In some cases, the average $|REE|$ can be reduced of more than 10% compared to estimations obtained without transformations. Another important observation is that eliminating all the TIVs is not the best solution. Indeed, we obtained the best results with weak transformations, *e.g.* $n = 1.5$ or $n = 1.25$. Applying a stronger transformation generates a TMD matrix that contains less TIVs, and, consequently, a TED matrix that contains generally smaller estimation errors. Nevertheless, TIVs are not the only source of estimation errors and, even if the TMD matrix contains no TIV, the TED matrix contains unavoidable estimation errors. Moreover, if we apply the transformation $y = x^{1/n}$, we established that the relative estimation error is multiplied by n when the ED matrix is computed from the TED matrix. Thus, for strong transformations (*i.e.* big values of n), small errors in the TED matrix become large errors in the ED matrix.

On the other hand, unlike stated in [82], this approach does not generate ED matrices that are exact reproductions (TIVs included) of the corresponding MD matrix. At least, we observed that it is not possible with non-linear transformations of the type $y = x^{1/n}$, and, obviously, not with a square root as stated in [82]. To generate an ED matrix which is an exact reproduction of the MD matrix, it is necessary to generate a TMD matrix that contains no TIV and to obtain a TED matrix without estimation errors. Since the TIVs are not the only source of estimation errors, the probability to obtain a TED matrix without estimation errors is very small. However, other types of transformation functions could provide better results than $y = x^{1/n}$. But we did not investigate it further.

Another problem with non-linear transformations is that they are quite difficult to parameterize in practice. Indeed, we have seen that they allow us to improve the accuracy of the estimations but the transformation to apply vary from one topology to the other. For example, with Planetlab and Meridian, we obtained the best results with $n = 1.5$. But, with P2PSim, $n = 1.5$ generates worse estimations than those obtained with a vanilla Vivaldi and the best results are obtained with $n = 1.25$. Moreover, the improvement vary significantly from one topology to the others: with Planetlab we observed a big improvement, with Meridian a significant improvement and, with P2PSim, a very small improvement. On other data sets it could even generate worse estimations than a vanilla Vivaldi. All of this makes a general estimation mechanism based on non-linear transformations difficult to parameterize and to justify in practice.

Finally, form the point of view of shortcuts detection, we have seen that simply searching shortcuts in the ED matrix (SDC) gives worse detection results than those obtained when our criteria are applied with the estimations produced by a vanilla Vivaldi. However, since non-linear transformations slightly increase the quality of the estimations they may improve the quality of our detection results: we have seen that applying EDC with estimations generated using non-linear transformations can, in some cases, provide better shortcuts detection results than applying EDC with estimations generated by a vanilla Vivaldi. Nevertheless, as indicated above, a transformation function that is suitable for one topology is not necessary suitable for another topology and we were not able to obtain improvements for all our data sets. Thus, we stopped our investigations on that subject.

In conclusion, non-linear transformations are clearly not the magic solution to allow Vivaldi to deal with TIVs. Another solution to obtain an ED matrix which restore the TIVs existing in the MD matrix is to use an estimation mechanism that has been developed with the TIVs in mind (unlike Vivaldi). In the next chapter, we will try to use one of these estimation mechanisms: DMF.

Chapter 7

Decentralized matrix factorization

Abstract

In chapter 5, we have presented three detection criteria (EDC, ADC and HDC) that are able to find interesting shortcuts for lots of paths in networks. These criteria use the estimations provided by an ICS, namely Vivaldi, to detect the shortcuts. But, it is well known that shortcuts (i.e. TIVs) generate unavoidable estimation errors with a classical ICS like Vivaldi. Our criteria take this into account and, in section 5.9, we have investigated which characteristics the estimation errors must have in order to provide good detection results (i.e. to be able to detect good shortcuts for most of the paths). But another way to detect shortcuts using estimations is to have estimations where the shortcuts are accurately estimated. DMF [42, 40] is an ICS that has been designed to naturally deal with TIVs (unlike Vivaldi). In this chapter, we will investigate if we can obtain better detection results with DMF than with Vivaldi.

7.1 Network delay prediction by matrix factorization

In the previous chapter, we investigated a way to allow Vivaldi to deal with TIVs. In the current chapter, instead of focusing on Vivaldi, we will observe the results obtained with an ICS, namely DMF, that has been developed to naturally deal with TIVs. We have already introduced DMF in chapter 3. Unlike Vivaldi and lots of other ICSes, DMF does not try to embed the delays into a metric space. With DMF, the problem of network distance prediction is seen as a matrix completion problem: the delay matrix contains the results of the measurements done between the nodes and their neighbors but lots of the delays are not measured and must be inferred from the measured ones. As shown in figure 7.1, DMF resolves the problem by matrix factorization.

Assume a network composed of n nodes. DMF builds a measured delay matrix $\mathcal{D}$ where some delays are measured (the delays between the nodes and their neighbors) and others are not. Then, it tries to fill in the holes existing in the matrix $\mathcal{D}$ in order to build a full estimated matrix $\hat{\mathcal{D}}$. The main observation about delay matrices is that they can generally be approximated by matrices of low rank [79]. Intuitively, for two near nodes, the delays to the other nodes are similar because, for a given destination, these nodes probably use the same Internet path. Thus, the lines corresponding to these nodes in the delay matrix are correlated. In the light of this observation, if $\hat{\mathcal{D}}$ is of low rank r, then it can be factorized into a product of two smaller matrices $\hat{\mathcal{D}} = \mathcal{X}\mathcal{Y}^T$. Thus, DMF's objective is to build $\mathcal{X}$ and $\mathcal{Y}$ from the information provided in $\mathcal{D}$.

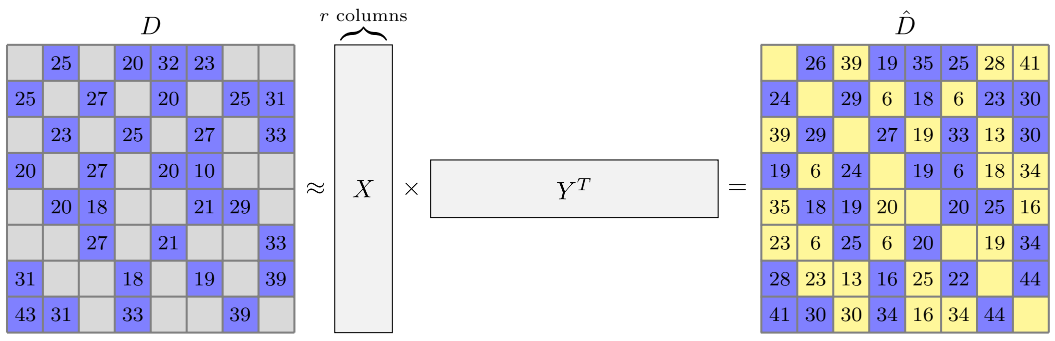


Figure 7.1: **Network delay prediction by matrix factorization.** Since $\hat{\mathcal{D}}$ is of low rank r, the objective is to use the information provided in $\mathcal{D}$ in order to compute two small matrices $\mathcal{X}$ and $\mathcal{Y}$ from which it is possible to compute $\hat{\mathcal{D}}$.

7.1.1 Overview

In DMF, $\mathcal{D}$ denotes the measured matrix and $\hat{\mathcal{D}}$ denotes the estimated matrix. $\mathcal{D}$'s and $\hat{\mathcal{D}}$'s elements are respectively denoted by d_{ij} and $\hat{d}_{ij}$. Obviously, DMF does not know the whole matrix $\mathcal{D}$. As in Vivaldi, each node performs measurements with a few other nodes (its neighbors) and only these values are known in $\mathcal{D}$. $\mathcal{W}$ denotes a matrix where $w_{ij} = 1$ if d_{ij} is known and $w_{ij} = 0$ otherwise. Given these notations, we can say that DMF tries to compute $\hat{\mathcal{D}}$ from $\mathcal{D}$ by minimizing the following function under the constraint that $\hat{\mathcal{D}}$ is of low rank r:

$$\sum_{i,j=1}^{n} w_{ij}\left(d_{ij} - \hat{d}_{ij}\right) \tag{7.1}$$

We already know that delay matrices are generally of low rank. Thus, a possible approach to build $\hat{\mathcal{D}}$ from $\mathcal{D}$ is to constrain the rank of $\hat{\mathcal{D}}$ (*e.g.*, $Rank(\hat{\mathcal{D}}) = r$) in order to have enough constraints to solve the matrix completion problem given in equation (7.1). However, solving such problem is difficult. To simplify the problem, DMF proposes to factorize $\hat{\mathcal{D}}$ into the product of two smaller matrices $\hat{\mathcal{D}} = \mathcal{X}\mathcal{Y}^T$. If $\hat{\mathcal{D}}$ is of rank r (with $r \lll n$), these matrices can be limited to r columns (and n lines). If $\hat{\mathcal{D}}$ is replaced by $\mathcal{X}\mathcal{Y}^T$ in equation (7.1), then DMF compute $\mathcal{X}$ and $\mathcal{Y}$ by minimizing the following function under the constraint that $\hat{\mathcal{D}}$ is of low rank r:

$$\sum_{i,j=1}^{n} w_{ij}\left(d_{ij} - x_i y_j^T\right) \tag{7.2}$$

where x_i (resp. y_i) is $\mathcal{X}$'s (resp. $\mathcal{Y}$'s) ith row and $x_i y_j^T = \hat{d}_{ij}$. If $\mathcal{D}$ were complete, solutions to the problem could be found by using singular value decomposition (SVD). With missing entries, the problem can be solved using iterative optimization methods such as Gradient Descent.

Matrix completion by matrix factorization suffers from a well-known problem called *overfitting* in the field of machine learning: the obtained $\hat{\mathcal{D}}$ matrix contains no or small errors on the $\hat{d}_{ij}$ for which d_{ij} is measured and large errors on the $\hat{d}_{ij}$ for which d_{ij} is unknown. To solve that problem, DMF introduces a regularization coefficient λ that penalizes the norms of the solutions. With that coefficient, DMF computes $\mathcal{X}$ and $\mathcal{Y}$ from $\mathcal{D}$ by minimizing the following function:

$$\sum_{i,j=1}^{n} \left(w_{ij} \left(d_{ij} - x_i y_j^T \right) \right) + \lambda \sum_{i=1}^{n} \left(x_i x_i^T \right) + \lambda \sum_{j=1}^{n} \left(y_j y_j^T \right) \tag{7.3}$$

To solve equation (7.3), we need information about the measurements performed by all the nodes. Thus, a centralized algorithm seems required to solve the problem. However, as DMF's "D" stands for "decentralized", DMF obviously proposes a decentralized resolution of equation (7.3).

In DMF, each node has two "coordinates": the x coordinate and the y coordinate. For node i, the x coordinate is x_i (*i.e.* $\mathcal{X}$'s ith line) and the y coordinate is y_i (*i.e.* $\mathcal{Y}$'s ith line). Like in Vivaldi, each node computes locally its own coordinates. To compute x_i and y_i locally, node i probes and gets the coordinates of m other nodes in the network called its *neighbors*. Thus, DMF's basic behaviour is exactly the same as Vivaldi's behaviour. Using these information, DMF computes x_i and y_i by minimizing the following functions:

$$\sum_{j=1}^{n} \left(w_j^i (d_{ij} - x_i y_j^T) \right) + \lambda x_i x_i^T \tag{7.4}$$

$$\sum_{j=1}^{n} \left(w_j^i (d_{ji} - x_j y_i^T) \right) + \lambda y_i y_i^T \tag{7.5}$$

where w_j^i represents the neighboring relationship of node j to node i, *i.e.* $w_j^i = 1$ if node j is a neighbor of node i and $w_j^i = 0$ otherwise. Thus, DMF addresses the large-scale optimization problem defined in equation (7.3) by decomposing it into a number of subproblems in equations (7.4) and (7.5). These subproblems can be solved locally at each node by using only local measurements.

A first version of DMF [42] proposed to find solutions to equations (7.4) and (7.5) using Alternating Least Squares (ALS). The algorithm performs correctly and converges quickly to accurate coordinates but it has a significant drawback: it requires each node to do measurements with its k neighbors simultaneously to update the coordinates. Thus, even if ALS performs well in simulations, it is impractical when deployed in real applications. In [40], a new version of DMF has been proposed under the name DMFSGD. This version is based on Stochastic Gradient Descent (SGD) instead of ALS. Unlike ALS, SGD processes measurements one by one and one at a time. This makes the system more flexible. We do not describe here how ALS or SGD can find solutions to equations (7.4) and (7.5). The interested reader can refer to [42] and [40] to find more information on the subject.

7.1.2 Advantages of DMF

Compared to Vivaldi and other classical ICSes, DMF has many advantages. In this section, we will investigate a few of them that can be useful when the estimation mechanism is used to find routing shortcuts.

Capacity to deal with TIVs

In our case, DMF's main advantage is its capacity to deal with TIVs. Indeed, since there is no embedding in a metric space, DMF does not require that the triangle inequality holds for the delays. In other words, DMF can provide an estimated matrix where TIVs are correctly estimated. Thus, we can potentially find routing shortcuts using SDC (see chapter 6) instead of using more sophisticated criteria like EDC, ADC or HDC. However, as Vivaldi used with non-linear transformations, DMF *can* provide estimations where TIVs are correctly estimated but, currently, we have no guarantee that this is the case. We will check this in section 7.2.

Capacity to estimate one way delays

In Vivaldi, the estimation between two nodes is the distance in the metric space between the coordinates of the nodes. Consequently, the estimation obtained for the path AB is exactly the same as the estimation obtained for the path BA. Thus, it is impossible to estimate metrics like one-way delays with Vivaldi because the one-way delay measured between the node A and the node B can differ from the one-way delay measured between the node B and the A. With DMF, d_{ij} must not necessary be equal to d_{ji}. The only major constraint with DMF is that the measured matrices must be approximately of low rank. We already know that this is the case for RTTs matrices, but [42] and [40] state that this is also the case with one-way delay matrices.

Capacity to estimate other metrics than delays

If the only major constraint with DMF is that the matrices must be of low rank, then it can estimate one-way delays, but it should also be able to estimate other metrics. For example, Liao *et al.* showed in [41] that it is possible to use DMF to obtain rough bandwidth estimations. With Vivaldi, estimating metrics like bandwidth is difficult to imagine. Indeed, intuitively, estimating delays by distances in a space seems logical since the physical distance between two nodes is an important component of the delay between these nodes. But it is absolutely not the case with metrics like bandwidth. In fact, we do not know if Vivaldi is able to estimate such metrics. We have not tried to do that. But, to our knowledge, there exists no popular results on the subject: some people proposed to do that in their future work but nobody seems to have proposed interesting results.

7.1.3 Parameters of DMF

Like with Vivaldi, we have to choose values for the parameters if we want to use DMF to conduct experiments. DMF has mainly three parameters: the regularization coefficient λ, the rank r and the number of neighbors m used by each node to compute its coordinates.

Rank

Intuitively, r corresponds to the number of unknown variables in each coordinate. In Vivaldi, it corresponds to the number of dimensions of the embedding space. In [40], Liao *et al.* observed that they obtained better results with values of r smaller or equal to 10. They also indicate that larger values of r accentuate the overfitting phenomena. Since we ran our experiments with Vivaldi in a 10-dimensional Euclidean space we choose to run our experiments with DMF using $r = 10$.

Regularization coefficient

As stated above, the regularization coefficient λ reduces the overfitting problem. The designers of DMF have observed that using $\lambda = 1$ is a good choice most of the time. However, with big values of r, a bigger value of λ is also necessary to avoid overfitting problems. Since we choose to use $r = 10$, using $\lambda = 1$ is good choice. Note that using $\lambda = 1$ and $r = 10$ is DMF's default configuration but it is not guaranteed to be the optimal choice for every situation. They have just observed empirically that this parameter setting leads to a good prediction accuracy for a large variety of data.

Number of neighbors

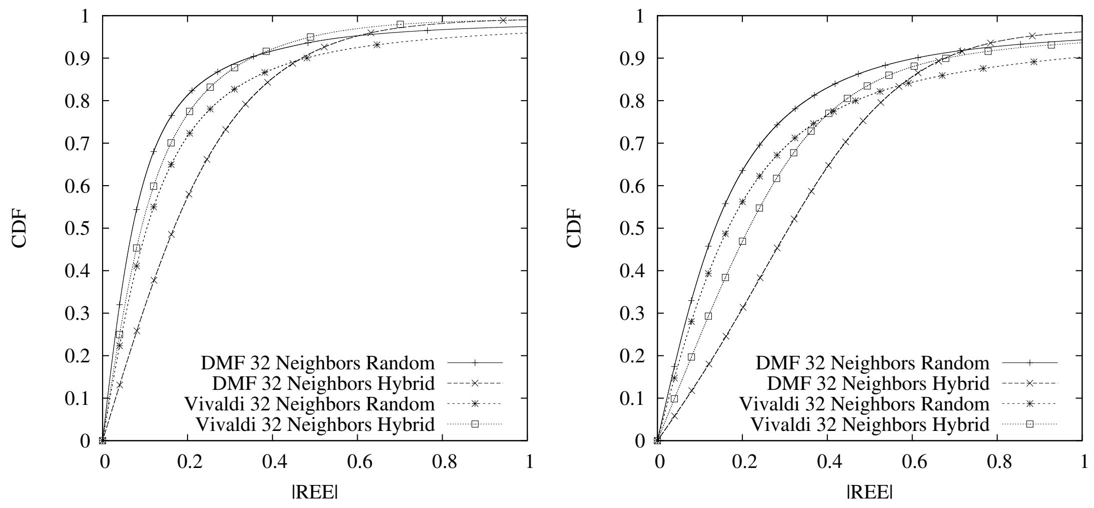
Intuitively, the number of neighbors m corresponds to the amount of data that is used to estimate each unknown variable in a coordinate. In [40], Liao *et al.* state that, according to the theory of matrix completion, using a value of m proportional to $r \times log(n)$ guarantees a decent prediction accuracy. Following this statement, using $m = 32$ like with Vivaldi seems a good choice for DMF. Indeed, with the P2PSim data set, $r \times log(n) = 32.41$ and, with the Meridian data set, $r \times log(n) = 33.98$.

Even if $r = 10$ and $m = 32$ seem to correspond to Vivaldi's 10 dimensions and 32 neighbors, DMF will consume more resources than Vivaldi with these parameters. Indeed, like in Vivaldi each node will perform measurements and exchange 10-dimensional coordinates with 32 neighbors. But, in DMF each node has two coordinates. Thus, the communication traffic is multiplied by two compared to the traffic generated by Vivaldi.

Neighbors selection scheme

In section 5.9, we have seen that, for ADC and HDC, using a hybrid neighbors selection scheme is the best choice to obtain good detection results. With Vivaldi, a hybrid neighbors selection scheme was also the best choice to obtain accurate estimations. Since this choice improved both the accuracy of the estimations and the accuracy of the detection of shortcuts, it can be recommended to the users.

With DMF, things are different. Intuitively, using systematically the $m/2$ nearest nodes as neighbors for every node will provide a significant amount of redundant information that is not necessary for matrix completion while useful information is missed. Indeed, if B is A's nearest node, there is a strong probability that A belongs to the set of B's $m/2$ nearest nodes. Thus, A will use B as neighbor and B will use A as neighbor, and they will both measure $RTT(A, B)$. Thus, more useful information is available to solve matrix completion problems when the measurements are randomly distributed in the matrix (especially when the available measurements are sparse, like in our situation). Nevertheless, we tested DMF with random neighbors and with hybrid neighbors. The CDF's of the $|REE|$ obtained with the different neighbors selection schemes are given in figure 7.2.



(a) P2PSim (b) Meridian

Figure 7.2: CDF of the absolute REEs obtained using DMF and different neighbors selection schemes.

Figure 7.2(a) gives the CDFs of the absolute REEs obtained with the P2PSim data set. In this figure, we see clearly that Vivaldi applied with hybrid neighbors gives better results than Vivaldi applied with randomly selected neighbors: for example, 77% of the paths have an $|REE|$ smaller than 0.2 with hybrid neighbors, while only 71% of the paths have an $|REE|$ smaller than that value with randomly selected neighbors. We see also that DMF applied with randomly selected neighbors provides more accurate estimations than Vivaldi applied with hybrid nieghbors. The last observation is that DMF applied with hybrid neighbors is the estimation mechanism that provides the worst estimations. For example, when DMF is applied with randomly selected neighbors, 81% of the paths have an $|REE|$ smaller than 0.2 while only 57% of the paths have an $|REE|$ smaller than that value when DMF is applied with hybrid neighbors. In figure 7.2(b) we see that the conclusions are similar with Meridian. With that data set, it is difficult to decide what is the best choice between a hybrid neighbors selection scheme and a random neighbors selection scheme for Vivaldi. This is probably due to the fact that most of the paths are small paths in Meridian. Thus, selecting random neighbors automatically leads to select near nodes and using a hybrid neighbors selection scheme is less critical for this data set. Nevertheless, even if the neighbors selection scheme is not critical with Vivaldi, it remains important with DMF: DMF with random neighbors still provides the best estimations while DMF with hybrid neighbors still provides the worst estimations. This indicates that having well-distributed measurements in the matrix to be completed is crucial for DMF.

In conclusion, there is a big difference between Vivaldi and DMF: with DMF the neighbors selection scheme that is the most suitable for our shortcut detection criteria is not the most suitable for the estimation mechanism. In section 7.2, we will try to detect shortcuts using both neighbors selection schemes but we must keep in mind that we cannot recommend the users to run DMF with hybrid neighbors[1].

7.2 Shortcut detection using DMF estimations

7.2.1 Error analysis

From section 5.9, we know that a good estimation mechanism for our criteria must minimize the over-estimation of the small paths and, for ADC and HDC, it must also use hybrid neighbors. We already know that using hybrid neighbors leads to bad estimations with DMF. Thus, using ADC and HDC with DMF seems difficult. In this section, we will evaluate if DMF tends to over-estimate the small paths or not in order to know if our detection criteria will be efficient using these estimations.

Like in section 5.9.4, we divide the whole range of RTTs in the P2PSim (resp. Meridian) data set into equal bins of 50 ms each. Then, for each bin, we compute the percentage of paths that are over-estimated (see figure 7.3(a)) and the average of the $|REE|$ observed for bin's paths that are over-estimated (see figure 7.3(b)). Since our criteria require an estimation mechanism that minimizes the over-estimation of the small paths, we must have as less as possible over-estimated small paths (*i.e.* the values in figure 7.3(a) must be as small as possible for the first bins) and, for the over-estimated small paths, the over-estimation must be as small as possible (*i.e.* the values in figure 7.3(b) must be as small as possible for the first bins).

In figure 7.3, dotted lines depict the results obtained with the Meridian data set and the solid lines depict the results obtained with P2PSim data set. Let us begin with the analysis of the percentage of paths that are over-estimated. In figure 7.3(a), we see that, on both data sets, DMF used with random neighbors over-estimates more the small paths than Vivaldi used with hybrid neighbors. For instance, with the P2PSim data set, the percentage of over-estimated paths in the first bin is equal to 64% when Vivaldi is used with hybrid neighbors while it is equal to 81% when DMF is used with random neighbors. Thus, DMF used with random neighbors has not the most suitable neighbors selection scheme for our criteria and it tends to over-estimate the small paths. It is not promising for our shortcuts detection results. Nevertheless, figure 7.3(a) indicates that DMF used with hybrid neighbors can potentially provide better detection results than those obtained

[1]Remember that the ICS is independent from our shortcuts detection mechanism and its estimations can be used by other applications. Thus, it must provide as accurate estimations as possible.

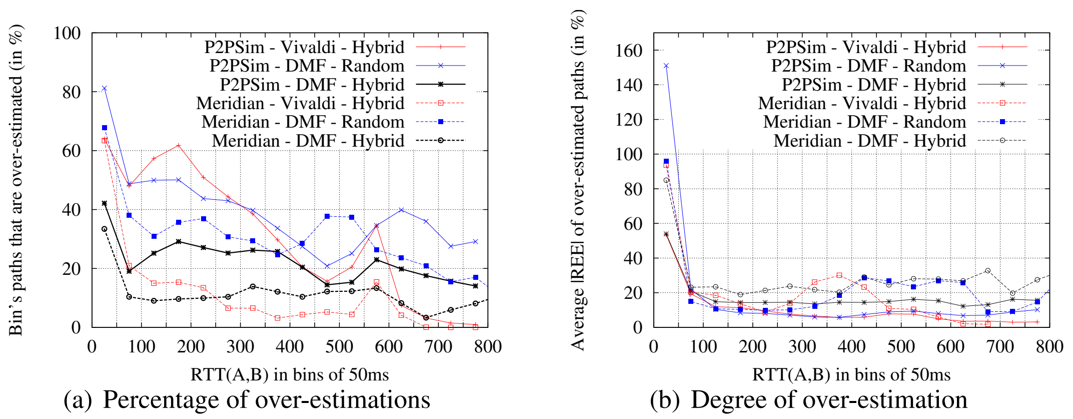


Figure 7.3: Over-estimations of the paths obtained with DMF and, either a random, or a hybrid neighbors selection scheme. Results obtained with Vivaldi and a hybrid neighbors selection scheme are also given to allow comparison with figure 5.9 page 102.

when Vivaldi is used with hybrid neighbors. Indeed, on both data sets, the percentage of over-estimated paths when DMF is used with hybrid neighbors (given by the bold lines in the figure) is always the smallest (at least for the first bins). For example, with the Meridian data set, the percentage of over-estimated paths in the first bin is equal to 63% when Vivaldi is used with hybrid neighbors while it is only equal to 34% when DMF is used with hybrid neighbors.

We will now have a look at the degree of over-estimation. For each bin, figure 7.3(b) gives the average $|REE|$ obtained for the over-estimated paths. On both data sets, in the first bins, we see that the average $|REE|$ of the over-estimated paths when DMF is used with hybrid neighbors is similar to the one obtained when Vivaldi is used with hybrid neighbors. A second observation is that DMF used with random neighbors over-estimates significantly the small paths. For example, with the P2PSim data set, the average $|REE|$ of the over-estimated paths belonging to the first bin is somewhere around 150% when DMF is used with random neighbors while it is equal to 53% when DMF is used with hybrid neighbors.

As conclusion, since its neighbors selection scheme is not suitable and, since it tends to over-estimate the small paths, DMF used with random neighbors should not give very good detection results with our detection criteria. Nevertheless, in the light of the results presented in this section, DMF used with hybrid neighbors should provide significantly better shortcuts detection results than those obtained when Vivaldi is used with hybrid neighbors.

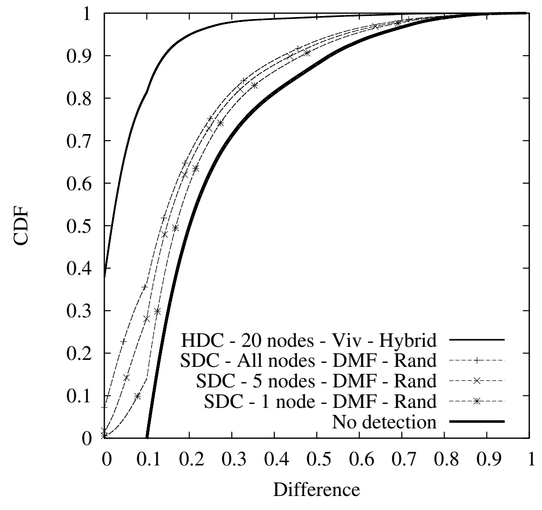
7.2.2 DMF with randomly selected neighbors

In the current section, we try to detect one-hop routing shortcuts using the estimations provided by DMF applied with randomly selected neighbors. Since DMF is able to provide estimations containing TIVs, we will first try to search shortcuts among these estimations (*i.e.* we will try to apply the SDC criterion described in chapter 6) before searching shortcuts using our detection criteria.

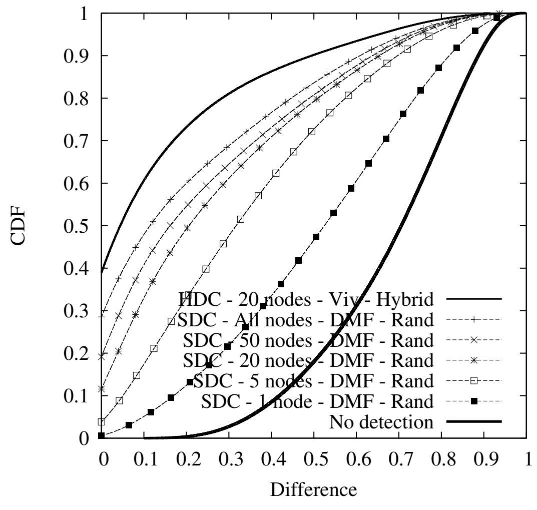
To evaluate the quality of detection results obtained using a given criterion we will compute the potential improvement[2] remaining after the detection process as we did in chapter 5. In the sequel of the current section, DMF denotes "DMF applied with randomly selected neighbors" and Vivaldi denotes "Vivaldi applied with hybrid neighbors".

Searching shortcuts among the estimations

Figure 7.4 gives the CDFs of the potential improvement remaining after the detection process for different values of the parameter k when we use SDC as detection criterion with the estimations provided by DMF. Figure 7.4(a) gives these results for the P2PSim data set and figure 7.4(b) gives these results for the Meridian data set.



(a) P2PSim



(b) Meridian

Figure 7.4: SDC applied with DMF used with random neighbors: Difference of G_r between the best and the best detected shortcuts.

On both data sets, we see that, even if we consider all the shortcuts provided by the estimated matrix, the detection results obtained with SDC and DMF are worse than those obtained with HDC and Vivaldi. With P2PSim, the difference between the best result obtained with SDC and our reference point is large. With HDC and Vivaldi, around 95% of the paths have a potential improvement after the detection process which is smaller than 0.2 while, with SDC and DMF, only 65% of the path have a potential improvement after the detection process which is smaller than that value. With Meridian, DMF is able to restore more shortcuts than with P2PSim. Thus, the detection results obtained with SDC and DMF are closer of those obtained with HDC and Vivaldi. Anyway, HDC applied with $k = 20$ and Vivaldi's estimations remains our reference point.

[2]Remember that this indicator is computed, for a given path, as the difference between the relative gain provided by the best existing shortcut and the relative gain provided by the best detected shortcut. To illustrate the quality of the detection results on a given data set, we draw the CDF of these differences computed for every path for which there exists at least one interesting shortcut. The faster such CDF rises, the better it is.

Shortcut detection using our criteria

The detection results obtained with our different criteria using the estimations provided by DMF are given in figure 7.5. Let us start with EDC. With the P2PSim data set, we see in figure 7.5(a) that EDC with DMF gives slightly worse detection results than EDC with Vivaldi. Indeed, with DMF, even if we consider all the nodes detected as shortcuts by EDC (*i.e.* $k = \infty$), the potential improvement remaining after the detection is bigger than the one remaining when we apply EDC with $k = 50$ and Vivaldi. Obviously, if using EDC with DMF does not improve the detection results obtained using EDC with Vivaldi, these results are also worse than those obtained using HDC and Vivaldi. With the Meridian data set, the situation is quite different. With Meridian, we saw in section 7.2.1 that the characteristics of the DMF estimations are similar to the characteristics of the Vivaldi estimations. Thus, we concluded that the detection results obtained with these estimation mechanisms should be equivalent. In figure 7.5(b), we see that using EDC with DMF slightly improves the detection results obtained using EDC with Vivaldi. Indeed, the potential improvement remaining after the detection process when we use EDC with Vivaldi and $k = 50$ is similar to the potential improvement when we use EDC with DMF and $k = 20$. However, these results are still worse than our reference (*i.e.* HDC with Vivaldi and $k = 20$).

Let us continue with the detection results obtained with ADC and HDC. Since the version of DMF used here does not use hybrid neighbors, we concluded in section 7.2.1 that ADC and HDC applied with DMF estimations should not provide good detection results. Figures 7.5(c) and 7.5(d) confirm this. With both data sets, we see that, when ADC is applied with $k = 50$, the detection results obtained with DMF are significantly worse than those obtained with Vivaldi. For HDC, figures 7.5(e) and 7.5(f) lead to the same conclusion if we compare the CDF obtained with $k = 20$.

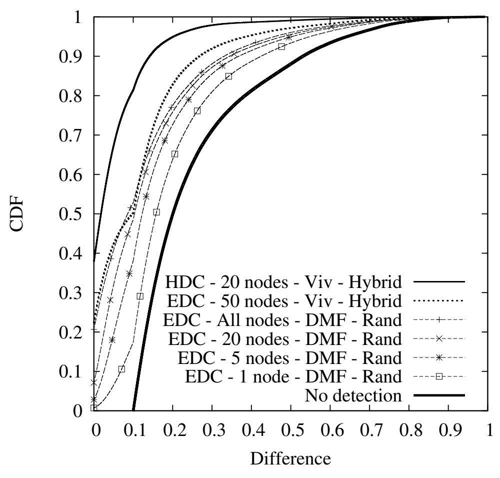
In conclusion, DMF applied with randomly selected neighbors fails to provide better detection results than those obtained with Vivaldi applied with hybrid neighbors. Our best detection results remains those obtained with HDC, $k = 20$ and Vivaldi estimations. In the next section, we will investigate if using DMF applied with hybrid neighbors improves these detection results.

7.2.3 DMF with hybrid neighbors

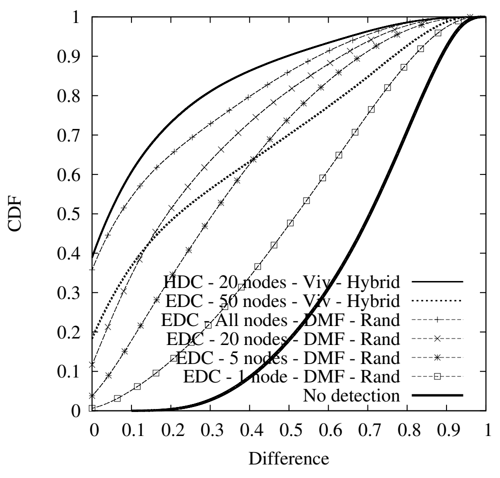
To analyse the quality of the detection results obtained when DMF is applied with hybrid neighbors, as usual, we will compute the potential improvement remaining after the detection process. Before trying to apply our criteria EDC, ADC and HDC, we will try to detect shortcuts by using the TIVs existing in the estimated matrix. In the sequel of the current section, DMF denotes "DMF applied with hybrid neighbors" and Vivaldi denotes "Vivaldi applied with hybrid neighbors".

Searching shortcuts among the estimations

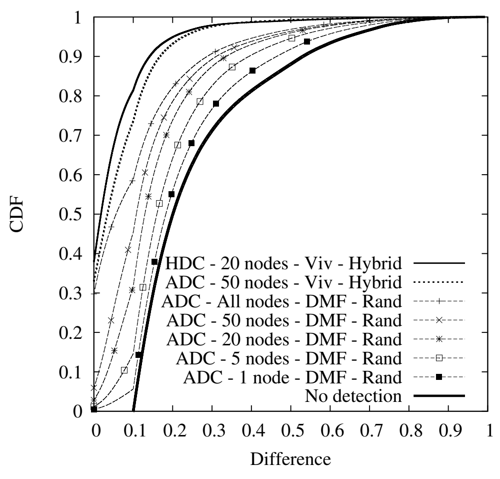
Figure 7.6 gives the potential improvement remaining after the detection process when the criterion SDC is used.



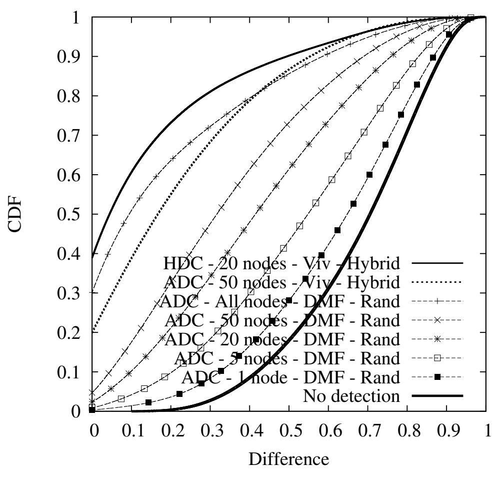
(a) EDC - P2PSim



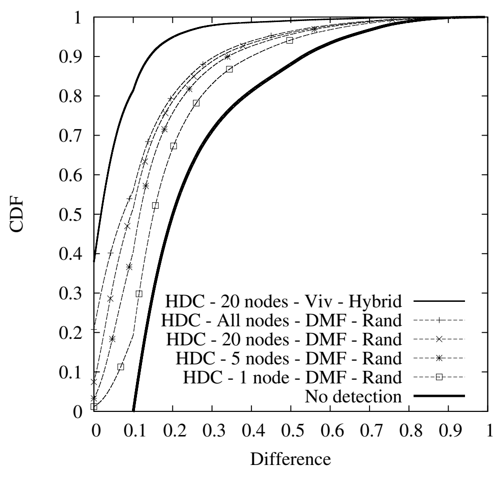
(b) EDC - Meridian



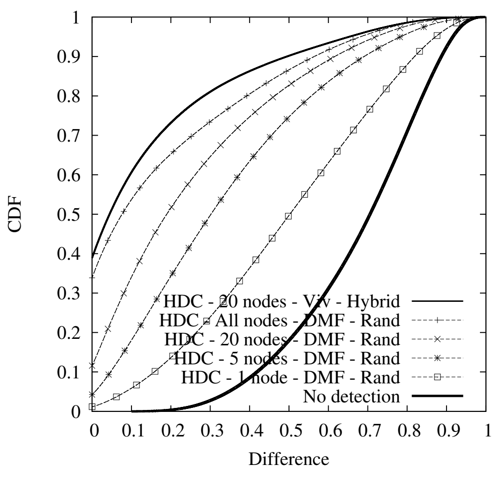
(c) ADC - P2PSim



(d) ADC - Meridian

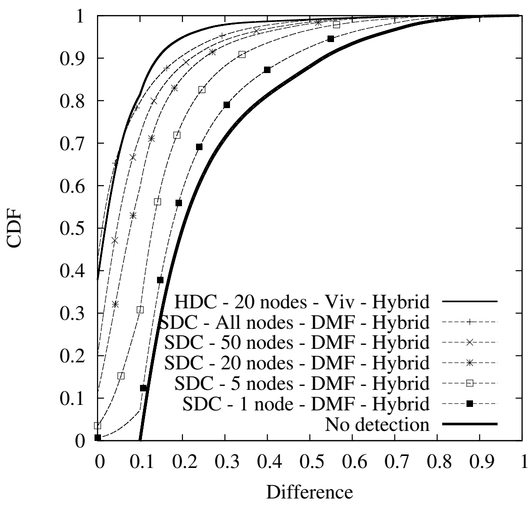


(e) HDC - P2PSim

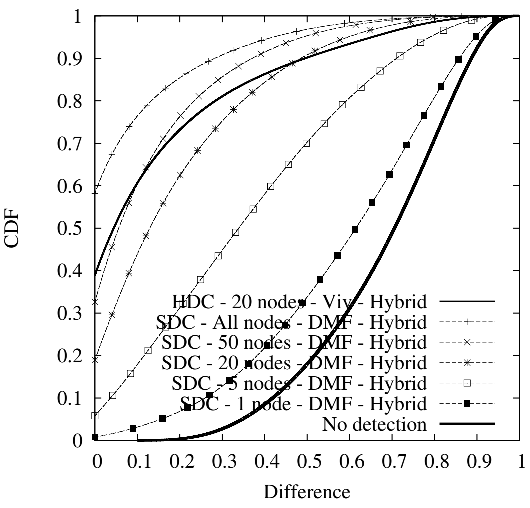


(f) HDC - Meridian

Figure 7.5: Difference of G_r between the best existing shortcut and the best detected shortcut with the criteria EDC, ADC and HDC using the estimations provided by DMF applied with randomly selected neighbors.



(a) P2PSim



(b) Meridian

Figure 7.6: SDC applied with DMF used with hybrid neighbors: Difference of G_r between the best and the best detected shortcuts.

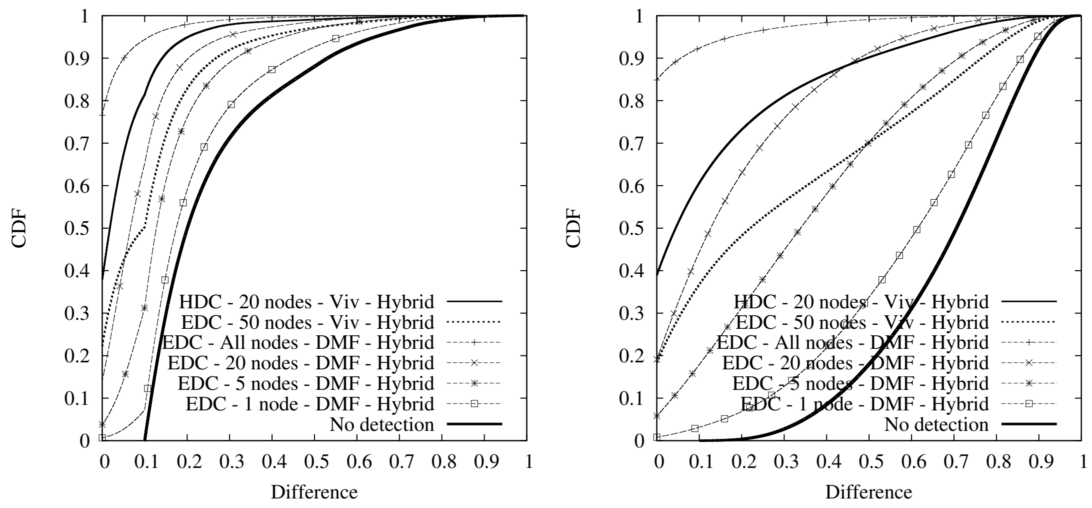
With the P2PSim data set, we see in figure 7.6(a) that DMF is able to reproduce a large part of the TIVs in its estimations: if we consider all the TIVs existing in the estimated matrix, the shortcuts detection results (represented by the CDF named "SDC - All nodes") are similar to those obtained when HDC is applied with $k = 20$ and Vivaldi's estimations. This is a quite good result. However, this indicates also that DMF does not reproduce all the shortcuts and, particularly, that it does not reproduce all the best shortcuts[3]. Moreover, the best shortcuts in the measured matrix do not always appear as the best shortcuts in the estimated matrix. For example, if we consider only the best shortcut with respect to the estimations for each path, the detection mechanism does not detect lots of interesting shortcuts. Indeed, if we look at the CDF named "SDC - 1 node", we see the potential improvement remains very important after the detection process. To obtain good detection results, we must consider large values of k. Anyway, with a same value of the parameter k (*e.g.* $k = 20$), we see that HDC applied with Vivaldi provides better detection results than SDC applied with DMF.

We have similar conclusions in figure 7.6(b) with the Meridian data set. Even if SDC detects interesting shortcuts (it works even better than on the P2PSim data set), HDC applied with $k = 20$ and Vivaldi's estimations remains our reference point. To obtain similar results when SDC is applied with DMF, we must consider $k = 50$ to obtain the same quality of detection results. This implies doing twice more measurements to find the best shortcuts among the candidates. That is not negligible.

Searching shortcuts using EDC

The detection results obtained when we use EDC to detect shortcuts with DMF estimations are given in figure 7.7.

[3]If this were the case, the potential improvement after the detection should be equal to 0 for every path.



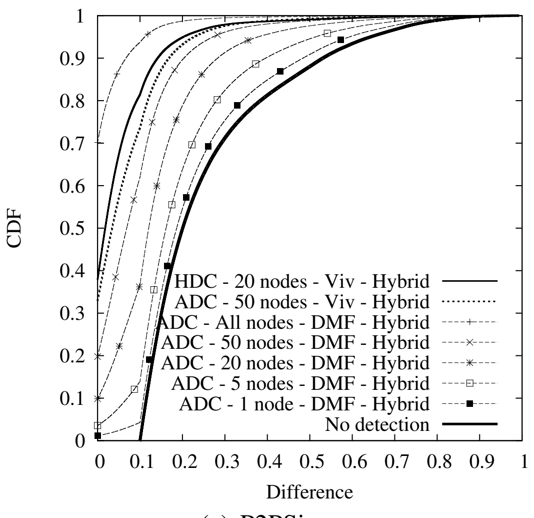
(a) P2PSim (b) Meridian

Figure 7.7: EDC applied with DMF used with hybrid neighbors: Difference of G_r between the best and the best detected shortcuts.

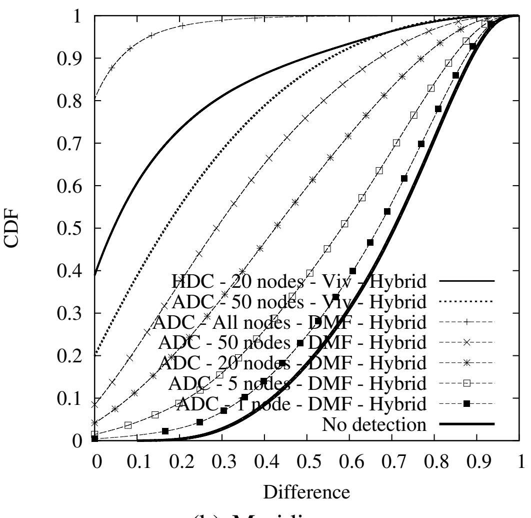
The fact that DMF does not over-estimate the small paths seems to be really a good thing for EDC. If we consider all the candidates detected by EDC (CDFs named "EDC - All nodes"), we see that the potential improvement remaining after the detection process is very small for lots of paths. For example, with P2PSim, EDC is able to find the best shortcut for more than 75% of the paths and the potential improvement remaining after the detection process is smaller than 0.1 for 95% of the paths. However, these very good detection results are obtained when we consider $k = \infty$. If we take small values for k, the detection results are not as good. But we can see that EDC applied with DMF estimations gives better results than EDC applied with Vivaldi estimations. Indeed, on both data sets, using EDC with $k = 20$ and DMF estimations provides better detection results than using EDC with $k = 50$ and Vivaldi estimations. Nevertheless, even if replacing Vivaldi by DMF improves the detection results obtained with EDC, this modification does not allow to beat the results obtained when HDC is used with $k = 20$ and Vivaldi estimations. Indeed, on both data sets, we see that the CDF obtained when EDC is applied with $k = 20$ and DMF estimations is worse than the CDF obtained when HDC is applied with $k = 20$ and Vivaldi estimations. Thus, HDC applied with $k = 20$ and Vivaldi estimations remains our reference.

Searching shortcuts using ADC

Figure 7.8 illustrates the quality of the detection results when ADC is used. The conclusion is roughly the same as when EDC is used. Indeed, if we consider all the nodes detected by ADC (*i.e.* if we use $k = \infty$), we obtain very good detection results. But, if we limit k to small values the results are mitigated and they are worse than what we obtain when HDC is applied with $k = 20$ and Vivaldi estimations.



(a) P2PSim



(b) Meridian

Figure 7.8: ADC applied with DMF used with hybrid neighbors: Difference of G_r between the best and the best detected shortcuts.

Even though we concluded in section 7.2.1 that DMF applied with hybrid neighbors generates estimations having suitable characteristics for our detection criteria, with ADC, the results obtained with DMF estimations are slightly worse than those obtained with Vivaldi estimations (see the curves named "ADC - 50 nodes" for both estimation mechanisms). Since considering all the nodes detected by the criterion allows us to find the best shortcut for most of the paths, the results presented here confirm that DMF estimations allow ADC to detect the best shortcuts (*i.e.* the best shortcuts are generally candidates). But, the problem is that most of the paths (not only the small ones) are under-estimated with large estimation errors. Consequently, the best shortcuts are not necessarily at the beginning of the ranking and, thus, using small values of k, ADC applied with DMF estimations fails to detect one of the best shortcuts for lots of paths. In section 5.3.3, we have already seen that ADC has more difficulties than EDC to rank the candidates. Thus, even though the large estimation errors do not prevent EDC from providing better results with DMF estimations than with Vivaldi estimations, they amplify ADC's ranking difficulties.

Searching shortcuts using HDC

The detection results obtained using HDC to detect shortcuts with DMF estimations are given in figure 7.9. Following the results presented in section 7.2.1, with DMF, HDC should be roughly equivalent to ADC. Indeed, since lots of the paths are under-estimated, in most of the cases HDC will be able to find one of A's (resp. B's) neighbors that is sufficiently near C with respect to the threshold. Thus, HDC will switch to EDC only a few cases and it will essentially behave like ADC. Figure 7.9 confirms that: the CDFs are slightly better than the CDFs obtained with ADC in figure 7.8. However, this improvement is small and the quality of the detection results are far from what we can obtain by using HDC with Vivaldi estimations.

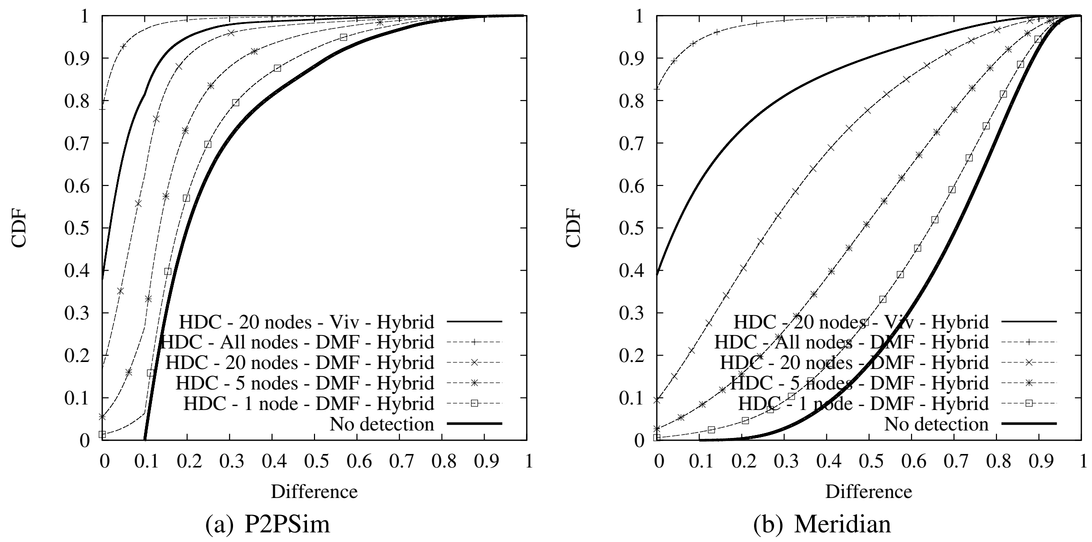


Figure 7.9: HDC applied with DMF used with hybrid neighbors: Difference of G_r between the best and the best detected shortcuts.

7.3 Conclusion

In this chapter we investigated the possibility to use the DMF estimations to detect shortcuts. In theory, this estimation mechanism is promising because it has been developed to deal with TIVs and it provides better estimations than Vivaldi. In practice, this estimation mechanism is not suitable for our shortcut detection criteria. The main problem is that such an estimation mechanism requires random neighbors to provide good estimations. Thus, it is not suitable for ADC and HDC. The second problem is that it has a tendency to over-estimate the small paths more than Vivaldi (at least, on the P2PSim data set). Thus, it is not suitable for EDC either. We tried anyway to apply our detection criteria using estimations provided by DMF applied with random neighbors and the results were systematically worse than what we obtain using the Vivaldi estimations.

We tried to run DMF with hybrid neighbors but we faced another problem. With this neighbor selection scheme, DMF does significant estimation errors and has a significant tendency to under-estimate all the paths. In theory, an estimation mechanism that under-estimates the small paths is good for our detection criteria: it allows our criteria to detect the best shortcuts and to consider them as potential shortcuts. However, with a hybrid neighbors selection scheme, DMF under-estimates significantly the major part of the paths. Thus, a large amount of nodes are considered as potential shortcuts. If we add the estimation errors to this, it is easy to understand that our criteria have difficulties to accurately rank the potential shortcuts. Thus, the criteria require big values of k to detect the best shortcut in most of the cases and the results are finally worse than those obtained with Vivaldi.

Notice that DMF applied with hybrid neighbors allows us to improve the results obtained with EDC compared to those obtained with Vivaldi. Even if the detection results obtained with EDC and DMF are worse than the results obtained with HDC and Vivaldi, these results could finally be considered as the best. Indeed, HDC is a little bit more complex than EDC mainly because it requires that the nodes exchange the results of the measurements they perform with their neighbors. Thus, if we consider the additional communication cost required by HDC, EDC applied with DMF estimations could finally be a better choice than HDC applied with Vivaldi estimations. However, DMF applied with hybrid neighbors has a major drawback: it is less accurate than DMF applied with randomly selected neighbors (and, than Vivaldi applied with hybrid neighbors). Since the coordinate system is an external component of our shortcut detection mechanism, its estimations could be used by other applications and they must naturally be as accurate as possible. Thus, imposing this estimation mechanism to all the applications is impossible. In conclusion, for the sequel of this thesis, HDC applied with Vivaldi estimations remains our reference result.

In the next chapter, we will try to use an estimation mechanism that reduces the estimation errors of the small paths (required for our criteria) and the global estimation error (required to be suitable for other applications). This estimation mechanism is our hierarchical version of Vivaldi.

Chapter 8

Hierarchical Vivaldi

Abstract

In chapter 4, we have seen that routing policies or path inflation can give rise to violations of the triangle inequality with respect to RTTs in the Internet. With Internet coordinate systems like Vivaldi, such TIVs introduce inaccuracy, as nodes in this particular case cannot be embedded into a metric space. Thus, we studied the TIVs existing in the Internet and we observed that the short paths are generally not involved in severe TIVs. In the light of this observation, we propose a Two-Tier architecture (opposed to a flat structure) of Vivaldi that mitigates the effect of TIVs on the quality of the estimations. These results have been published in [7, 29, 6] and we will summarize them in the current chapter. Since the Two-Tier Vivaldi tends to reduce the estimation errors on the small paths compared to a flat Vivaldi, these estimations should be suitable for our shortcut detection criteria. We will investigate this in the last part of the chapter.

8.1 Two-Tier Vivaldi

In section 4.3.3, we have observed that small triangles are less often (severe) TIVs than big triangles. In summary, we observed that few severe TIVs are found in triangles whose bases are smaller than $100\,ms$ in the P2PSim data set and smaller than $60\,ms$ in the Meridian data set. These observations motivate our hierarchical approach of Vivaldi. We intend to create clusters whose diameters do not exceed too much $100\,ms$ (resp. $60\,ms$) for the P2PSim data set (resp. the Meridian data set). Then, we will run different local Vivaldi instances limited to the nodes belonging to a same cluster. Since we may expect much fewer severe TIVs in each cluster, the hierarchical Vivaldi is likely to improve the accuracy of intra-cluster estimations.

8.1.1 Overview

Since small triangles are less often (severe) TIVs, any 3 edges with small RTTs should not violate too much the triangle inequality rule and should be more easy to embed in a metric space. Put simply, short paths are more embeddable than long paths that tend to create severe TIVs. We exploit such property to limit the impact of TIVs on coordinates, by proposing a Two-Tier Vivaldi approach. The main idea is to divide the set of nodes into clusters and to run an independent Vivaldi in each cluster. Clusters are composed of nodes within a given coverage distance.

Since Vivaldi instances running on each cluster are independent, nodes are collecting latency information from only a few other neighbors located within the same cluster. In this way, coordinates of nodes belonging to different clusters cannot be used to estimate the RTT between these nodes. We keep then running a Vivaldi system at a higher level (i.e. the whole set of nodes) in order to ensure that coordinates computed between any two nodes belonging to different clusters 'make sense'. Coordinates computed at the lower (resp. higher) level of the architecture are called *local coordinates* (resp. *global coordinates*).

Each node need not necessary belong to a cluster. A node that does not belong to a cluster just takes part in the higher level of the Two-Tier Vivaldi and just computes its global coordinate (GC). On the other hand, nodes belonging to a cluster have to compute two coordinates: the global coordinate (GC) and the local coordinate (LC). The way the coordinates are used to compute estimations is illustrated in figure 8.1. To compute an estimation between two nodes belonging to the same cluster (*e.g.*, node 1 and node 2 in figure 8.1), we use the local coordinates. To compute an estimation between nodes belonging to different clusters (*e.g.*, node 3 and node 5 in figure 8.1) or between a node that does not belong to any cluster and any other node (*e.g.*, node 4 and node 6 in figure 8.1), we use the global coordinates.

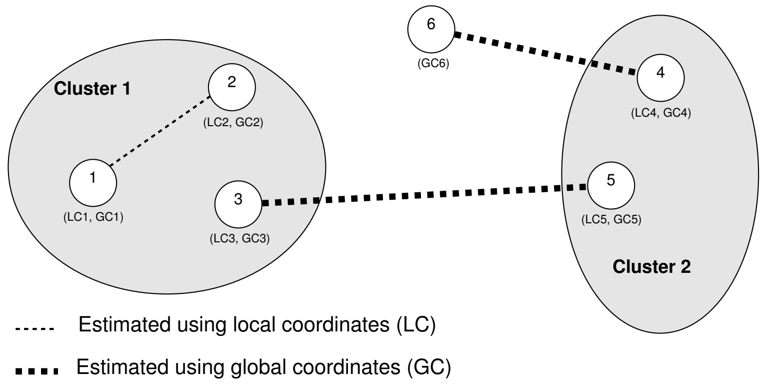


Figure 8.1: Usage of the coordinates to compute estimations with the Two-Tier Vivaldi.

An important thing to understand is that the higher level of the architecture is a classical "flat" Vivaldi. Indeed, every node takes part in the higher level and can choose any node as neighbor to compute its global coordinate. Thus, global coordinates will provide the same accuracy as the coordinates computed using an ordinary Vivaldi (*i.e.* non-hierarchical). At the lower level, each node that belongs to a cluster participates in a Vivaldi limited to its cluster. Since there should be no, or few TIVs between the nodes belonging to a given cluster, the embedding of the delays in a metric space should be easier for the local Vivaldi running in that cluster. Thus, local coordinates should provide more accurate estimations for intra-cluster distances than a flat Vivaldi. In summary, our Two-Tier Vivaldi should improve the accuracy of the estimations of small paths (*i.e.*, intra-cluster paths) without degrading the accuracy of the other paths.

Next, we describe how we set up the clusters we experimented with to illustrate our results on the Two-Tier Vivaldi structure. We built the clusters manually for our experiments but a self-organized clustering scheme will be presented in section 8.2.

8.1.2 Clustering method

We have based our clustering method on the coordinates obtained by running a flat Vivaldi. As already shown by Dabek *et al.* [15], 2-dimensional Euclidean coordinates lead to five major clusters of nodes for the P2PSim data set. We took the three most populated clusters as our main clusters. Nodes that have not been selected in any cluster are then considered at the higher level of the architecture only, and they compute their global coordinate only. However, this first step in our cluster selection is only based on the estimated RTTs (as predicted by the flat Vivaldi) and not on the actual RTTs between nodes. In fact, according to their relative errors, some nodes can be misplaced by the flat Vivaldi and some clusters could have a larger diameter than expected. Our second step has then consisted in a cross-checking of our preliminary clustering using the delay matrix. We proceed in a recursive way as follows: for any two nodes belonging to the same cluster, the cluster constraint is that the distance between these two nodes should be smaller than the cluster diameter. We then begin by verifying this cluster constraint, testing it on all the pairs of each cluster according to their actual distance provided by the delay matrix. Afterwards, we remove the top node that causes more violations of the constraint[1]. We then recursively check the cluster constraint until no more pairs inside the cluster violates the cluster constraint. Following this clusters selection method, we obtained the clusters described in figure 8.2(a) for the P2PSim data set.

Even if our tests have been mainly done using the P2PSim data set, clusters can also be built for the Meridian data set. Indeed, for the Meridian data set, 2-dimensional Vivaldi's coordinates give also three major clusters and the clusters obtained after the removal of the misplaced nodes are given in figure 8.2(b).

	Nodes	Diameter
Cluster 1	565	140 ms
Cluster 2	169	100 ms
Cluster 3	93	60 ms

(a) P2PSim data set

	Nodes	Diameter
Cluster 1	560	80 ms
Cluster 2	563	80 ms
Cluster 3	282	70 ms

(b) Meridian data set

Figure 8.2: Characteristics of the clusters used for our experiments

[1]This node is likely to have a high embedding error

8.1.3 Two-Tier Vivaldi algorithm

As presented in section 8.1.1, the Two-Tier Vivaldi is composed of completely independent instances of Vivaldi running in the clusters and one instance of Vivaldi running at the higher level of the architecture. It can be run this way but this will generate useless additional costs compared to a classical Vivaldi. Indeed, if it works this way, the Two-Tier Vivaldi will generate about twice more measurements than the flat Vivaldi because each node belonging to a cluster will do measurements with m neighbors to compute its local coordinate and with m other neighbors to compute its global coordinate. Thus, simply running two instances of Vivaldi (see algorithm 2 page 33) with different sets of neighbors on each node is not an optimal solution.

A simple solution to reduce the measurement costs of the Two-Tier Vivaldi is to reduce the number of neighbors used for the local Vivaldi instances. Indeed, in section 3.3.4, we have seen that $m = 32$ neighbors are necessary to obtain accurate estimations with large topologies like P2PSim (1740 nodes) and Meridian (2500 nodes). But the clusters where the local instances of Vivaldi are running contain less nodes: our biggest cluster contains only about 570 nodes. Thus, with such smaller topologies, working with less neighbors should not degrade the accuracy of the estimations. We did the test with the P2PSim cluster 1 and we observed no difference between the quality of the estimations obtained with $m = 16$ and the quality of the estimations obtained with $m = 32$. However, decreasing m to 8 gives significantly worse estimations. Thus, if the Vivaldi instance running at the higher level of our architecture works with $m = 32$ neighbors, we can run the Vivaldi instances of the lower level of our architecture with $m/2$ neighbors.

This simple approach allows us to reduce the costs induced by the Two-Tier Vivaldi but it remains more costly than the flat Vivaldi. This additional cost is due to the fact that each node uses two distinct sets of neighbors to compute its local coordinate and its global coordinate. To have the same measurement cost for the Two-Tier Vivaldi and for the flat Vivaldi, each node must use the same set of neighbors for the two coordinates. Since, the local coordinate requires less neighbors than the global coordinate, there a simple way to reach such result. Indeed, the $m/2$ neighbors used to compute the local coordinate are not sufficient to compute the global coordinate (it requires also neighbors outside the cluster) but they can be used for this purpose. Thus, if each node computes its global coordinate using the $m/2$ "local" neighbors and $m/2$ additional neighbors (not necessarily chosen inside the cluster), the Two-Tier Vivaldi can compute two coordinates with a unique set of m neighbors. This way, the Two-Tier Vivaldi has the same measurement cost as Vivaldi.

Consequently, instead of running two distinct instances of the Vivaldi algorithm on each node, we will run one instance of a specific 2tier-Vivaldi algorithm on each node. With the 2tier-Vivaldi algorithm, the node has two coordinates: its local coordinate $\vec{l}$ and its global coordinate $\vec{g}$. Like in the Vivaldi algorithm, the node computes the confidence it has in its coordinate through an estimation of its local error. Since the node has two coordinates, it has two error estimations: el, the error for the local coordinate and eg, the error for the global coordinate. Finally, each node must know the identifier of the cluster it belongs to. This information is stored in a variable c. If the node does not belong to a cluster, its variable c is equal to -1 and its local coordinate is not used (as well as the corresponding error el). Each time a node performs a measurement with one of its neighbors, it receives the neighbor's cluster ID and the neighbor's local and global coordinates (and the corresponding errors el and gl). With these information, the procedure presented in algorithm 5 can be run.

Algorithm 5 2tier-Vivaldi (rtt, $\vec{l}_{remote}$, $\vec{g}_{remote}$, el_{remote}, eg_{remote}, c_{remote})

```
Require: A measurement has been done between the node and one of its neighbors
 1: {$c_{local}$ is the node's cluster ID (-1 if it does not belong to a cluster)}
 2: {$c_{remote}$ is the neighbors's cluster ID (-1 if it does not belong to a cluster)}
 3: {$\vec{l}_{local}$ is the local coordinate of the local node}
 4: {$\vec{g}_{local}$ is the global coordinate of the local node}
 5: {$\vec{l}_{remote}$ is the local coordinate of the neighbor}
 6: {$\vec{g}_{remote}$ is the global coordinate of the neighbor}
 7: {$el_{local}$ is the node's local error on the local coordinate}
 8: {$eg_{local}$ is the node's local error on the global coordinate}
 9: {$el_{remote}$ is the neighbor's error on the local coordinate}
10: {$eg_{remote}$ is the neighbor's error on the global coordinate}
11: {$rtt$ is the measured RTT}
12: $w = eg_{local}/(eg_{local} + eg_{remote})$
13: $est = ||\vec{g}_{local} - \vec{g}_{remote}||$
14: $e_s = |rtt - est|/rtt$
15: $eg_{local} = e_s \times (ce \times w) + eg_{local} \times (1 - (ce \times w))$
16: $\delta = cc \times w$
17: $int = rtt - est$
18: $dir = u(\vec{g}_{local} - \vec{g}_{remote})$
19: $f = dir \times int$
20: $\vec{g}_{local} = \vec{g}_{local} + \delta \times f$
21: if $((c_{local} \neq -1) \wedge (c_{local} == c_{remote}))$ then
22:     $w = el_{local}/(el_{local} + el_{remote})$
23:     $est = ||\vec{l}_{local} - \vec{l}_{remote}||$
24:     $e_s = |rtt - est|/rtt$
25:     $el_{local} = e_s \times (ce \times w) + el_{local} \times (1 - (ce \times w))$
26:     $\delta = cc \times w$
27:     $int = rtt - est$
28:     $dir = u(\vec{l}_{local} - \vec{l}_{remote})$
29:     $f = dir \times int$
30:     $\vec{l}_{local} = \vec{l}_{local} + \delta \times f$
31: end if
```

This procedure is very similar to the procedure presented in algorithm 2 page 33. Whatever the neighbor is, we update the global coordinate (between line 12 and line 20). The operations are exactly the same as in algorithm 2. Then, if the node and its neighbor belong to the same cluster, we also update the local coordinate (between line 22 and line 30) using the same operations as in algorithm 2.

8.1.4 Neighbors selection

In section 8.1.3, we decided that each node must choose $m/2$ neighbors inside its cluster and $m/2$ other nodes (that can be inside or outside its cluster). The m neighbors are used to compute the global coordinate and the local coordinate is computed only with the first $m/2$ neighbors. In the current section, we define exactly how the neighbors are chosen.

We know that Vivaldi provides better estimations with a hybrid neighbors selection scheme than with randomly selected neighbors. With the Two-Tier Vivaldi, we have also a natural tendency to choose hybrid neighbors. Indeed, the $m/2$ neighbors chosen in the cluster are, by definition, near nodes (but not necessarily the nearest), while the $m/2$ other neighbors can be randomly chosen in order to obtain something similar to a hybrid neighbors selection scheme. Thus, with the Two-Tier Vivaldi, we can legitimately ask if it is necessary to work with a complicated hybrid neighbors selection scheme, while simply choosing randomly $m/2$ neighbors in the cluster and $m/2$ neighbors in the whole set of nodes should generate a similar result. We investigate this using the P2PSim data set. We worked with $m = 32$ neighbors and we considered two neighbors selection schemes:

1. A *hybrid* neighbors selection scheme that works as follows:
   (a) For a node that belongs to a cluster:
      - Take the 8 nearest nodes inside the cluster
      - Choose randomly 8 other nodes inside the cluster
      - Choose randomly 16 nodes among those that are not already selected

   (b) For a node that does not belong to any cluster:
      - Take the 16 nearest nodes
      - Choose randomly 16 nodes among those that are not already selected
2. A *random* neighbors selection scheme that works as follows:
   (a) For a node that belongs to a cluster:
      - Choose randomly 16 nodes inside the cluster
      - Choose randomly 16 nodes among those that are not already selected

   (b) For a node that does not belong to any cluster:
      - Choose randomly 32 nodes

We applied Two-Tier Vivaldi on the P2PSim data set with the clusters defined in figure 8.2(a). Figure 8.3 gives the CDFs of the $|REE|$ obtained with both neighbors selection schemes. The main observation on that figure is that Two-Tier Vivaldi applied with hybrid neighbors performs significantly better than Two-Tier Vivaldi applied with randomly selected neighbors. Thus, using the hybrid scheme is really important to obtain good estimations. In the light of this observation, we will use a hybrid neighbors selection scheme in the sequel.

In figure 8.3, we can also see that the Two-Tier Vivaldi applied with randomly selected neighbors performs significantly better than Vivaldi applied with random neighbors. That is logical since a Two-Tier Vivaldi with random neighbors is somewhat similar to a flat Vivaldi with a hybrid scheme. If we compare Two-Tier Vivaldi applied with hybrid neighbors to Vivaldi applied with hybrid neighbors, there is an improvement but it is quite

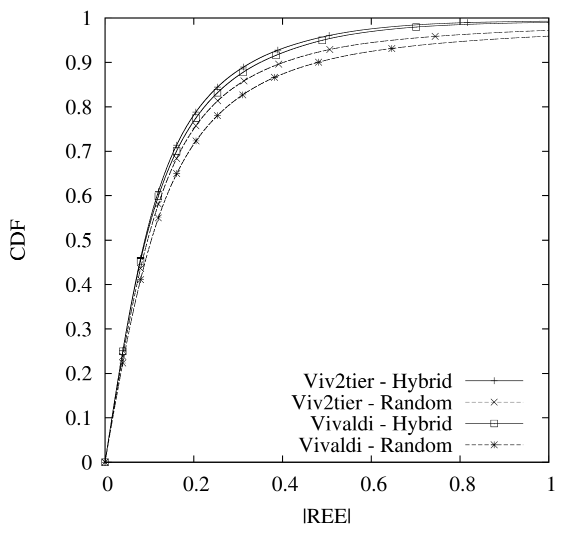


Figure 8.3: Comparison of neighbors selection schemes for the Two-Tier Vivaldi. The experiments have been carried out with the P2PSim data set and the clusters defined in figure 8.2(a)

small. This happens because the Two-Tier Vivaldi improves only a small part of the total number of estimations. Indeed, it can only improve the estimations of intra-cluster paths. For the P2PSim data set, with the values given in figure 8.2(a), we have:

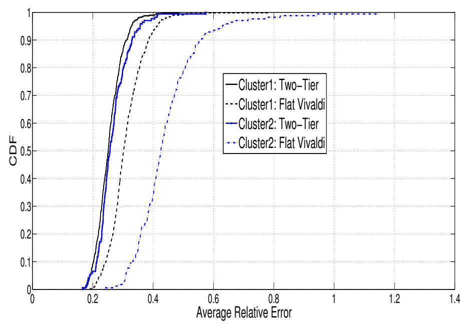
$$\begin{aligned}
&\text{Number of paths in cluster 1} = \frac{565 \times 564}{2} = 159\,330\\
&\text{Number of paths in cluster 2} = \frac{169 \times 168}{2} = 14\,196\\
&\text{Number of paths in cluster 3} = \frac{93 \times 92}{2} = 4\,278\\
\Rightarrow\ &\text{Total number of intra-cluster paths} = 159\,330 + 14\,196 + 4\,278 = 177\,804
\end{aligned}$$

$$\begin{aligned}
&\text{Total number of paths} = \frac{1740 \times 1739}{2} = 1\,512\,930\\
\Rightarrow\ &\text{Percentage of paths that are intra-cluster paths} = \frac{177\,804 \times 100}{1\,512\,930} = 11.75\%
\end{aligned}$$

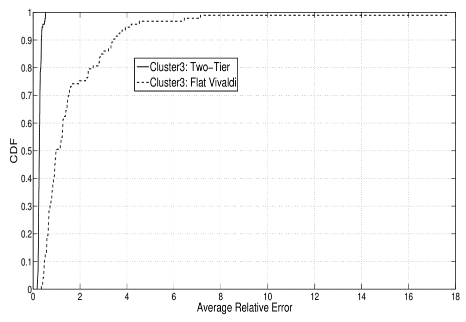
Thus, the Two-Tier Vivaldi can only improve the estimations obtained for 11.75% of the paths in the P2PSim data set. Consequently, the global improvement compared to Vivaldi estimations cannot be huge. That is why, in the next section, we will evaluate the performance of Two-Tier Vivaldi by focusing on the intra-cluster paths.

8.1.5 Performance evaluation of Two-Tier Vivaldi

To evaluate it, we ran Two-Tier Vivaldi on our data sets using a 2-dimensionalEuclidean space, hybrid neighbors and the clusters defined in figure 8.2. First, we use the relative error as our main performance indicator. We compute the ARE (Average Relative Error - see section 4.4 page 58) over all nodes to represent the accuracy of the overall system.



(a) Cluster 1 and cluster 2



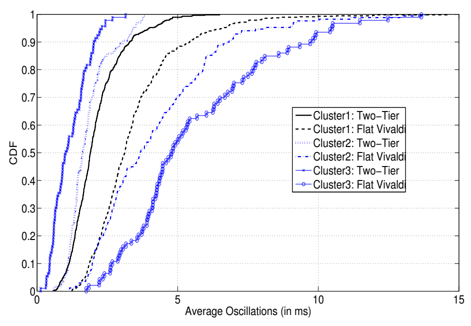
(b) Cluster 3

Figure 8.4: Comparison of relative errors for P2PSim data set using a 2-dimensional space: Flat Vivaldi versus Two-Tier Vivaldi.

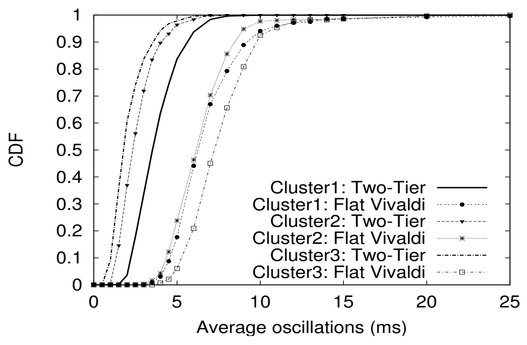
Figures 8.4(a) and 8.4(b) represent the CDFs of the ARE computed for the nodes belonging to our three clusters. We clearly see that the relative errors computed using the local coordinates are smaller than the relative errors computed using Vivaldi coordinates. For instance, in cluster 2, more than 90% of the nodes have an ARE smaller than 0.3 when they use the Two-Tier Vivaldi local coordinates to predict intra-cluster distances. With the Vivaldi coordinates, over half of the nodes belonging to cluster 2 have an ARE bigger than 0.5 for the estimation of intra-cluster paths. Worse cases (for Vivaldi) are observed with cluster 3. As depicted in figure 8.4(b), the flat Vivaldi system collapses with very high effective relative errors for more than 70% of the nodes. With the Two-Tier architecture, nodes are clearly performing much better. We observed the same trend for the Meridian data.

It is worth noticing here that cluster 3 is the smallest cluster in terms of diameter. The observation of the embedding relative errors in this cluster confirm then our findings related to the effect of edges lengths on the TIVs severity and thus on their impact on the embedding. More generally, improvements inside these clusters is explained by the fact that intra cluster nodes, when computing their local coordinates select only nearby nodes as their neighbors. This constraints the node-to-neighbor edge lengths and thus reduces the likelihood of selecting severe TIVs. When encountering severe TIVs that cause high absolute errors, a node updates its coordinate, by jumping back and forth across its actual position. When limited to TIVs of low absolute severity, a node converges 'smoothly' towards an approximation of its correct position, then would stick to such position, and oscillate much less. In essence, it gains confidence in its local error faster and performs more accurate embedding.

Limiting the neighborhood inside the cluster should then limit the high oscillations due to long and severe TIVs. In a second step, we then observed the coordinates' oscillation of nodes belonging to our three clusters. Like in chapter 4, we consider the average oscillation values as the average oscillations during the last 500 ticks of our simulations. We can observe that the three curves representing the CDF of the Two-Tier architecture oscillations are the highest one in figure 8.5(a) and figure 8.5(b). This clearly shows that local coordinates of nodes inside our clusters oscillate with less amplitude. For instance,



(a) P2PSim data set



(b) Meridian data set

Figure 8.5: CDF of coordinates oscillations for flat Vivaldi and Two-Tier Vivaldi.

in figure 8.5(a) more than 80% of the average oscillations are less than $3ms$ when nodes are computing local coordinates, whereas, only 40% of nodes in flat Vivaldi have, their oscillations less than this value. For Meridian data (figure 8.5(b)), nodes oscillate over large ranges. More than 50% of nodes in flat Vivaldi have their oscillations superior to $5\ ms$. Using the Two-Tier Vivaldi reduces this amplitude.

The results presented above have been computed before we decided to work with a 10-dimensional Euclidean space in order to avoid as much as possible prediction inaccuracies caused by the choice of the space (see section 3.3.3). However, working with 10 dimensions instead of working with 2 dimensions does not change the conclusions. Figure 8.6 shows that, for the P2PSim data set, intra-cluster paths are better estimated using Two-Tier Vivaldi's local coordinates than using flat Vivaldi's coordinates.

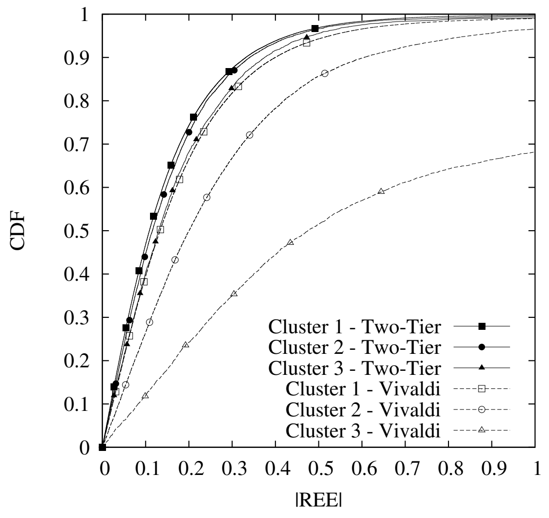


Figure 8.6: Comparison between Two-Tier Vivaldi and flat Vivaldi with a 10-dimensional space. The graphs give the CDFs of the $|REE|$ computed for intra-cluster paths. The experiment has been done with the P2PSim data set using the clusters defined in figure 8.2(a).

8.2 A self-organized clustering scheme

In section 8.1, we evaluated the performance of the Two-Tier Vivaldi using manually built clusters. To be usable, the Two-Tier Vivaldi requires a clustering scheme that is autonomous and distributed. In [6], we proposed a self-organized clustering scheme and we will summarize these results in the current section.

With the self-organized clustering scheme, our goal is twofold. Firstly, we address the problem of constructing efficient clusters in an autonomous way by building on an existing ICS system. Secondly, our clustering scheme aims at providing a self-managed clustering structure to overlay-based applications, to allow both topology awareness and scalability of these applications. The novelty of our approach lies in simultaneously relying on the partial knowledge of coordinates of nodes involved in ICS operations, and on a distributed clustering algorithm based on adaptive back-off strategy, to construct efficient topology-aware clusters in a load-balancing way. The main idea is to allow each node to identify a set of clusters in the network, using its own knowledge of a set of nodes' coordinates (as provided by the ICS in which it is involved), and to verify the validity of such clusters using a few measurements towards the identified cluster heads. The distributed algorithm is scheduled using an exponential back-off strategy, where nodes plan their own wake-up time to verify the existence of clusters in their proximity or not. Our main objective behind this strategy is to load balance the clustering process, while adapting to previous clusters creation, and hence optimize the maintenance and measurements overhead.

It is worth noticing that the notion of *awakening* used here is only related to the clustering algorithm. Indeed, in the Two-Tier architecture, every node participates permanently in the coordinate system for its coordinate(s) computation and, periodically, its clustering process *wakes up* to discover if it can join an existing cluster (or create a new one). Even though only the clustering process wakes up and goes back to sleep, for more simplicity, we will denote this as a *node awakening* in the sequel.

We provide two variants of our distributed algorithm: a first variant, called *Cooperative*, aims at reducing the expected time to construct clusters for the whole network. This approach induces some overhead to inform other nodes that they are likely to belong to a newly created cluster. A second *Selfish variant* is also introduced, where nodes are more selfish and can only form and/or join clusters when they wake up, without any assistance (or guidance) from other nodes that woke up earlier. In both cases nodes use only knowledge provided by a subset of other nodes, in some neighborhood as explained later, and obtain the needed pieces of information (coordinates, existing cluster heads) by piggybacking them in the messages exchanged by the ICS system.

8.2.1 QT (Quality Threshold) clustering algorithm

Clustering is defined as a process of partitioning a set of elements into a number of groups based on a measure of similarity between the data (distance-based approaches) or relying on the assumption that the data come from a known distribution (model-based approaches). For our self-clustering process, we aim at exploiting nodes' coordinates as a first approximation of the inter-node distances existing in the actual network topology.

As nodes' coordinates do not follow any a priori distribution, we will focus on distance-based clustering. Moreover, since we aim at providing a self-clustering process that is performed in a distributed way among all the nodes of the network, the optimal number of clusters that can be created is not known in advance. Approaches that set a constraint on the number of clusters to be formed (such as K-means, C-Means Fuzzy Clustering, etc.) are thus inappropriate.

Having in mind these facts, we choose to use the Quality Threshold algorithm (*QT-clustering*) to propose our self-organized clustering scheme. This algorithm has been initially proposed by Heyer *et al.* [25] for genetic sequence clustering. It is based on the unique constraint of the cluster diameter, as a user-defined parameter. For the QT-clustering and for the remainder of the current chapter we define the cluster diameter as the maximal distance existing among any two members of the cluster. The QT-clustering is an iterative algorithm and starts with a global set that includes all the elements (*e.g.* nodes coordinates) of the data set, and then returns a set of clusters that respects the quality threshold. Such threshold is defined in terms of the cluster diameter.

Initially, a candidate cluster seeded with the first element of the data set is formed. Then, elements are iteratively added to the cluster. Each iteration adds the element that minimizes the increase in cluster diameter. The process continues until no element can be added without surpassing the diameter threshold. A second candidate cluster is formed by starting with the second element of the data set and repeating the procedure. Note that all elements are made available to the second candidate cluster. That is, the elements from the first candidate cluster are not removed from consideration. The process continues for all elements. At the conclusion of this stage, we have a set of candidate clusters. The number of candidate clusters is equal to the number of elements, and many candidate clusters overlap. At this point, the largest candidate cluster is selected and retained. The elements it contains are removed from consideration and the entire procedure is repeated on the smaller set. A possible termination criterion is when the largest remaining cluster has fewer elements than some specified threshold.

8.2.2 Self-Clustering process

In this section we describe how we exploit the QT-clustering algorithm to provide a distributed self-organized clustering process, based on the knowledge of a subset of nodes' coordinates in a metric space, resulting from running a positioning system to estimate network distances. We will denote by (direct) neighbors the set of peer nodes that are used as neighbors in the ICS for the purpose of coordinate computation. We will also denote by *long-sight* neighbors, the union of these (direct) neighbors and the neighbors' neighbors (i.e., node's 2-hop neighbors). For instance, if a node has 32 neighbors in order to compute its coordinate, its long-sight neighbors will be formed by at most 1024 nodes.

Description

The general idea of our clustering algorithm is to distribute the clustering tasks among nodes in the network relying not only on measurements towards a potential existing cluster, but also on their knowledge of the coordinates of their long-sight neighbors. In other words, if a node wakes up and does not find directly an existing cluster it may belong to, it tries to construct such cluster based on the coordinates as provided by the ICS it is running. In such a way, nodes that do wake up earlier try to create clusters that their peers waking up later may join. Put simply, nodes perform trailblazing of the network conditions, to construct the clusters in a distributed way, while optimizing the needed overhead. Three main advantages could then be considered. Firstly, nodes do not need global knowledge of nodes in the network, nor distances between these nodes, nor a common landmark/anchor infrastructure. Secondly, the network is not overloaded by measurements performed to obtain the cluster structure. And, thirdly, the network is able to self-construct the clusters that may exist.

During the cluster forming phase, nodes are initially in a waiting mode. Each node waits for an initiator timer according to an exponential random distribution. The clustering process follows the procedure presented in Algorithm 6 and can be described as follows: each time a node wakes up, it gets the list of existing cluster heads in the network. Although such information could be obtained by requesting the set of long-sight neighbors that the node is aware of, we choose to perform this information retrieval by exploiting the communication already established at the level of the ICS. Existing cluster heads are propagated in the network by simply piggybacking in the classical ICS messages the identity of the cluster head(s) of cluster(s) a node belongs to. Considering these already existing clusters, each node verifies its membership to one of them. If the measurement towards the cluster head satisfies the cluster diameter, say D, meaning that such distance is less than $D/2$, the node simply joins such cluster by sending a "join" message to the cluster head. In section 8.1, we have seen that natural clusters for our data sets have diameters smaller than $140\,ms$. Following this observation, for our simulations, we set the upper bound of the cluster diameter D to $140\,ms$. Finding a way to adapt automatically this upper bound to the network is still one of our future work. Depending on the maximum number of clusters a node can join, say k, such "join" procedure could be repeated with other cluster heads.

Nevertheless if none of the distances to existing cluster heads satisfies the clustering criterion, the node starts the QT-clustering algorithm on the basis of the coordinates of its long-sight neighbors. It is worth noticing that this clustering is just a first approximation. Indeed coordinates may be subject to distance estimation errors, resulting from inaccuracies in coordinates. However this gives the node an approximate view of its neighbors positions, and in particular of the clusters that could be formed from this approximation. This first coordinate-based clustering phase allows the node to identify a set of clusters in the metric space of the ICS. This set of clusters is then subject to a verification according to direct measurements.

When a node has verified that its distance to an identified cluster head satisfies the clustering criterion, it decides to inform this potential cluster head that it should create a cluster, and waits for a confirmation. The cluster creation is conditioned by the acceptance of the requested cluster head. In fact, a potential cluster head could refuse to lead a cluster because of load constraints, or more specifically because its actual distance to an already existing cluster head has been considered too short. To this end, when a node is informed

Algorithm 6 Self-organized clustering: Procedure when a node wakes up

```
1:  if The node is already in at least one cluster then
2:      The node goes back to sleep;
3:  else
4:      The node gets the list of existing cluster heads known by its long-sight neighbors;
5:      The node measures RTTs to all existing cluster heads;
6:      Let $C$ be the list of existing cluster heads within a range $RTT < D/2$;
7:      if $C \neq \emptyset$ then
8:          The node joins at most the $k$ nearest clusters whose heads are in $C$;
9:      else
10:         Let $S$ be the list of coordinates of the node's long-sight (1-hop and 2-hop) neigh-
            bors;
11:         The node runs a QT-Clustering on $S \Rightarrow$ This returns a set of clusters;
12:         if The node is in none of these clusters then
13:             The node goes back to sleep;
14:         else
15:             The node selects a cluster head in its cluster;
16:             The node measures the RTT to this new potential cluster head;
17:             if $RTT > D/2$ then
18:                 The node goes back to sleep;
19:             else
20:                 The node freezes all of its long-sight neighbors (by sending them a mes-
                    sage);
21:                 if A neighbor answers that it is already frozen by another node then
22:                     The node goes back to sleep;
23:                 else
24:                     The node notifies the selected cluster head and waits for confirmation;
25:                     if Confirmation is positive then
26:                         The node joins the cluster;
27:                     else
28:                         The node goes back to sleep;
29:                     end if
30:                 end if
31:             end if
32:         end if
33:     end if
34: end if
```

that it is a potential cluster head, it measures its distance to the list of cluster heads it is aware of. If at least one of these distances is less than $\alpha \times D/2$, for some $1 < \alpha < 2$, distance between the two cluster heads is considered too short to construct a new cluster, and the request is refused. In this case, the node that identifies this cluster head is informed of this refusal and goes back to sleep. Otherwise, *i.e.* if the cluster is created, nodes that wake up later follow this decision and consider the cluster head among the list of existing cluster heads.

The algorithm relies on self-organization of nodes. When a node decides to join a cluster, two variants could drive the process of nodes joining the identified clusters.

The cooperative variant: The first variant, with the main goal of speeding up the clustering process, is to inform all identified nodes in a cluster of their potential membership to such cluster, and let them check this fact with direct measurements.

The selfish variant: The second variant trades off the speed of cluster creation against a reduced measurement overhead. In this case nodes never inform others that they may belong to a newly created cluster, and let them discover this fact when they wake up.

Finally, it is worth noticing that the wake-up procedure allows also some adaptation to changes in the network. Since distances in the network may evolve over time, including the distances of nodes towards their identified cluster head(s), a node should not stick to any cluster, and should also verify its membership to additional clusters due to new network conditions. Waking up from time to time, following the distributed scheduling as presented below, allows them to check their membership to existing clusters, and thereby adapt to changes in network conditions.

Distributed Scheduling of wake-up timers

During the cluster forming phase, nodes are initially in a waiting mode. Each node waits for an initiator timer according to an exponential random distribution, *i.e.*

$$f(t_i) = \lambda_i e^{-\lambda_i t_i}$$

where

$$\lambda_i = \lambda_0 \, \frac{n_i}{N_i}$$

n_i being the number of non already clustered nearby neighbors, and N_i being the total number of known long-sight nearby neighbors. By nearby nodes we refer to nodes whose coordinates indicate that they are (likely to be) within some specified range. To set the timer according to an exponential random distribution, we set $p_t = random(0, 1)$, compute λ_i as described above and let

$$t_i = \frac{-1}{\lambda_i} \, ln(1 - p_t)$$

The wake-up timer could then be computed as $timer = min(t_i, MAX\_Timer)$. From the expression of t_i it is obvious that the timer decreases when λ_i increases. Therefore such timer will ensure that the nodes with more residual non-clustered neighbors have more opportunities to (re)initiate the clustering algorithm, since their timer is more likely to elapse before other nodes. The main idea behind this exponential backoff scheduling is to load-balance the clustering process as initiated by nodes in the network, while optimizing the time needed to construct and join the clusters.

We can also expect that in one MAX_Timer period enough nodes initiate the clustering algorithm. To this end the selection of λ should satisfy the following inequation:

$$Prob(t > MAX\_Timer) \leq 1 - p$$

where p is the expected percentage of nodes initiating the algorithm. Therefore, in such a case

$$\int_{MAX\_Timer}^{+\infty} \mathrm{f}(t)\, dt \leq 1 - p \quad \Rightarrow \quad \lambda \geq -\left(\frac{ln(1-p)}{MAX\_Timer}\right) \tag{8.1}$$

Based on (8.1) we can calculate λ needed to ensure that a percentage of nodes initiate the algorithm, at least in the initial state, when nodes are not clustered yet. From (8.1) we can conclude that

$$\lambda_{min} = -\left(\frac{ln(1-p)}{MAX\_Timer}\right)$$

is sufficient to ensure this.

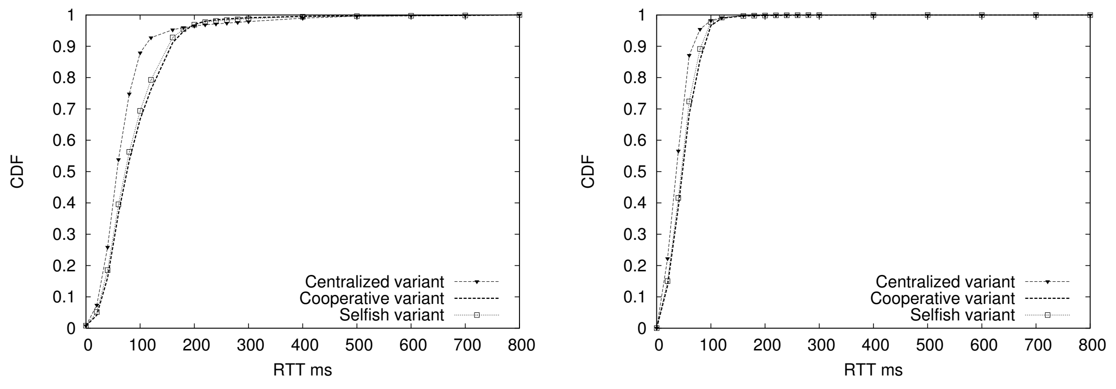
8.2.3 Analysing the Clusters

In this section we present the results of an extensive simulation study of the self-organized clustering process. We performed a set of simulations using the P2PSim data set and Meridian data set. In our simulations, we allowed nodes to join at most two clusters ($k = 2$) and we set the expected maximum cluster diameter D to 140ms for the P2PSim dataset and to 80ms for the Meridian dataset. The maximal timer for a sleeping node is set to 5 minutes and the minimum distance between any two cluster heads is fixed to $3/2 \times D/2$ (*i.e.* $\alpha = 3/2$). As we used coordinates as provided by an ICS in the first step of our self-organised clustering, we deployed the Vivaldi system as a prominent representative of purely P2P coordinate systems. Each node runs the Vivaldi system, setting the number of its neighbors to 32 and our results are obtained for a 2-dimensional coordinate space. We choose not to illustrate our results by using more accurate coordinates, for ease of deployment and low computing loads, at the cost of some loss in clusters accuracy.

We evaluate the performance of our clustering algorithm with respect to three main indicators. (i) The clusters quality: it is the deviation between the expected cluster diameter and the actual diameter. (ii) The convergence time: it is the time needed by our distributed algorithm to cluster 95% of the nodes in the system. This allows us to differentiate between the initial phase of the algorithm, when clusters are yet in the construction process, and the steady state, when nodes continue to manage their membership to already constructed clusters. Finally, we measured (iii) the overhead: it is the number of exchanged messages and the number of measurements performed. We can further split the overhead during the initial phase and during the steady state. We compare the two variants of our algorithm, and when needed we compare our distributed self-clustering algorithm to a centralized approach.

Clusters Quality

We can evaluate the cluster quality according to the deviation between the expected cluster diameter, as we set it in the QT-clustering and the actual diameter as obtained after our



(a) P2PSim dataset (b) Meridian dataset

Figure 8.7: CDF of the RTT of the intra-cluster paths.

self-organized clustering process reaches its steady state. However, the cluster size is also an important parameter we should mention. A cluster populated with only a few nodes, even though its diameter is optimal, may be of little use.

To evaluate the clustering quality in terms of cluster diameter, figures 8.7(a) and 8.7(b) show the Cumulative Distribution of the actual delays (RTTs) between the members of the same identified cluster (called intra-cluster RTTs). In other words, these figures show, for both data sets, the proportion of nodes that actually violate the diameter constraint. We compare the proportion of these violations for the two variants of our clustering algorithm and for a centralized approach. In this case, a centralized approach consists in emulating a centralized entity that collects the coordinates of all nodes in the system, computes in a centralized way clusters resulting from these coordinates using the QT-clustering and then informs all nodes of the identified cluster heads. These nodes verify their membership to these clusters, and join clusters if the diameter constraint with their cluster heads is verified. The main reason why we compare our algorithm to such a centralized algorithm is to evaluate how partial knowledge of neighborhood and coordinates impact our clustering performance.

Figure 8.7(a) shows that more than 85% of the intra-cluster links satisfy the cluster diameter constraint, with an RTT smaller than the expected diameter. The same trend is observed in figure 8.7(b) for the Meridian dataset, with more than 95% of clustered nodes scattered in delimited clusters, respecting the expected cluster diameter. We also note that both variants achieve the same performance, which is actually not surprising, since the main difference between our two variants is when nodes join a cluster, and not how they join it. The centralized approach creates slightly more accurate clusters. However, this little difference is overwhelmed by onerous cost induced by a centralized approach that needs global knowledge of both coordinates and nodes in the system.

Performing a QT-clustering based on coordinates of long-sight neighbors gives us a first approximation of nodes positioning. Even though nodes measure network distances, as RTTs, towards identified cluster heads, this does not prevent some mutual distances between cluster members to be above the expected diameter, due to TIVs. Using coordinates reduces the proportion of diameter violations, but since coordinates only provide distance estimations, errors may always exist. Example 7 illustrates the problem.

Example 7. *Consider a cluster head C and two nodes A and B. These nodes and distances between them are represented in figure 8.8. Suppose that the clustering process works with a diameter $D = 100\ ms$. If node A (resp. node B) wants to join C's cluster, it measures the RTT with the cluster head and obtains an RTT equal to $30\ ms$ (resp. $40\ ms$). Since that value is smaller than the diameter, the two nodes consider that they can join C's cluster. However, ACB is a TIV and $RTT(A, B)$ violates the diameter constraint. The problem is that a node does not perform maesurements with all the nodes in its cluster before joining it.*
Even during the QT-clustering phase, such problem can appear. Indeed, the clustering is based on estimations of the paths. In chapter 4, we have seen that TIV bases are generally under-estimated by Vivaldi. Thus the $160\ ms$ path between A and B can be estimated to a smaller value. For instance, if AC and CB are correctly estimated, then $60\ ms$ is a possible estimation for AB because the triangle inequality rule holds. With AB estimated at $60\ ms$, the QT-clustering algorithm wrongly considers that A and B can belong to the same cluster.

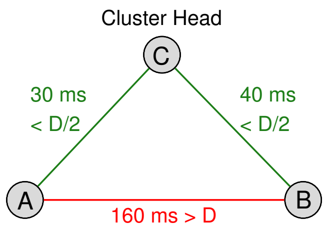


Figure 8.8: **Example of clustering inaccuracies caused by TIVs.** Nodes A and B consider both that they can join C's cluster. But, since ACB is a TIV, the path AB is longer than the diameter.

As shown in table 8.1, the number of clusters identified by our algorithm ranges from 9 to 11 for both variants. However, we can in both cases consider 3 main clusters, with an average population of 700 nodes each for the P2PSim dataset, and 1260 nodes as an average population of each cluster in the Meridian dataset. The percentage of nodes that have not been clustered is roughly 3.8%. The bottom part of table 8.1 will be presented later.

Convergence time

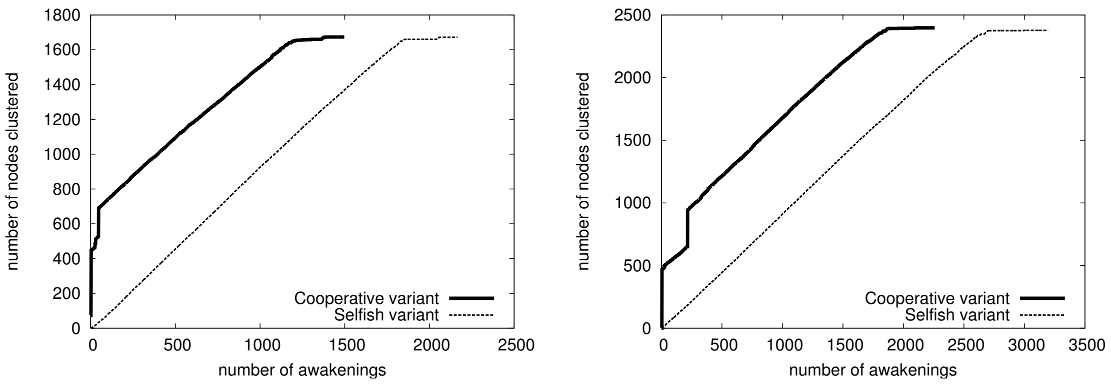
To separate the initial phase from steady state, we analyse the evolution of the number of clustered nodes versus the number of awakenings (and hence versus the number of clustering process calls) for both variants. As depicted in figures 8.9(a) and 8.9(b), the curves labeled "Selfish Variant" follow linear evolutions. Such observation is expected

	Cooperative Variant		Selfish Variant	
	P2PSim	Meridian	P2PSim	Meridian
Number of clusters	9	9	11	9
Number of unclustered nodes	67	81	68	102
Total Number of pings	11116	17003	20125	18075
Total Number of messages (excluding pings)	1582	2300	843	246
Convergence time (seconds)	1875	1658	1658	2300
Ping rate before convergence (pings/s)	4.48	8.05	9.95	6.45
Before convergence:				
Mean ping rate (pings/node$\times$s)	0.0026	0.003	0.0057	0.0026
Max ping rate (pings/node$\times$s)	0.027	0.038	0.0398	0.023
Mean msg rate (msg/node$\times$s)	0.0005	0.0006	0.0003	$4\,10^{-5}$
Max msg rate (msg/node$\times$s)	0.403	0.635	0.23	0.049
After convergence:				
Mean ping rate (pings/node$\times$s)	0.0002	0.0002	0.0002	0.0002
Max ping rate (pings/node$\times$s)	0.03	0.032	0.034	0.022
Mean msg rate (msg/node$\times$s)	10^{-6}	$6\,10^{-7}$	$3\,10^{-7}$	$6\,10^{-7}$
Max msg rate (msg/node$\times$s)	0.0007	0.0003	0.0002	0.0005

Table 8.1: Characteristics of the clustering process

since at most one node can join a cluster at each awakening, and then the growth of the number of clustered nodes can be at best linear. Clusters in the "Cooperative" variant may cumulate node membership with each node's awakening, because information of potential membership to a newly created cluster is sent by the creator of a cluster to identified members. In the curves labeled "Cooperative approach", we can then observe for both data sets the steps corresponding to a set of nodes joining simultaneously a defined cluster. Such steps allow this variant to cluster more than 95% of nodes in 1210 awakenings for the P2PSim dataset and in 1854 awakenings for the Meridian dataset, whereas the "Selfish" takes more occurrences, up to 2749 awakenings. Such faster clustering comes at the expense of costs of spreading information and exchanged messages, as we will discuss in the following section.

Let us consider that our system is in the steady state if at least 95% of existing nodes have been clustered. Although such parameter relies on our a priori knowledge of the number of nodes that cannot be clustered in the system, it gives us however a suitable way to separate the initial phase from the steady state. It is important to notice that the convergence time, although different in terms of number of awakenings, is roughly the same in real time (in seconds) spent to cluster 95% of the nodes, as shown in table 8.1. This confirms our choice of the exponential-backoff strategy to set timers. Remember that our algorithm will ensure that the greater the λ is, the lower the timer is, giving hence more opportunity to (re)initiate the clustering algorithm. This guarantees a higher probability to initiate the clustering algorithm in an "area" populated by nodes that have not been clustered yet. This gives a way to adjust the awakening rate according to the number of



(a) P2PSim dataset (b) Meridian dataset

Figure 8.9: Evolution of the number of clustered nodes.

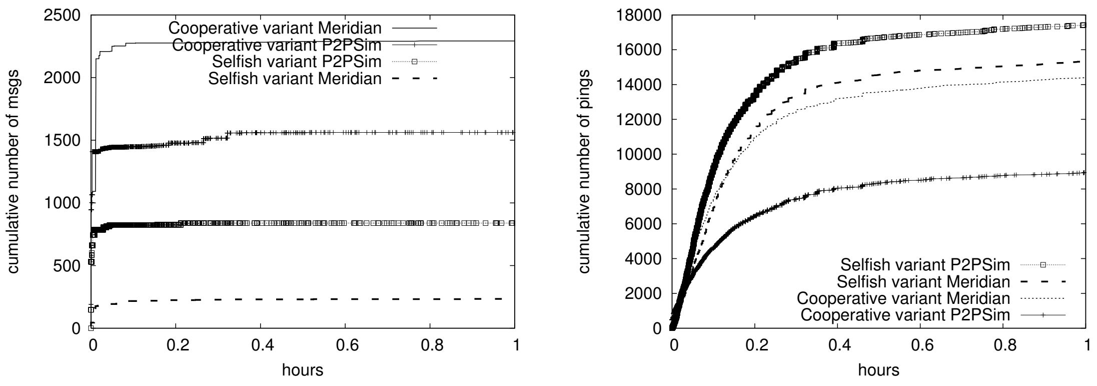
clustered nodes independently from the number of nodes existing in the system. Such trend is emphasized in the "Cooperative variant" where the convergence time is less than 2000 seconds for both data sets. Next we discuss the cost of the self-clustering algorithm.

Overhead

To observe the control messages and measurement overhead, we differentiate between the two states of the system: the initial phase (when clusters are built) and the steady state. We observe the induced overhead as the number of measurements performed, but also as the number of exchanged messages during the clustering process. Figure 8.10(a) depicts the cumulative number of exchanged messages versus time (up to one hour). We do not consider the "join" messages sent by clustered nodes to their cluster heads, but focus on the difference that may exist between the two variants of the algorithm.

The sharp rise of the curves in the initial phase is due to the fact that, at the beginning of the algorithm, nodes have the same probability to wake up, since all nearby neighbors are not clustered yet. We manage to resolve such potential conflictual situation using the "freeze" messages, sent to the long-sight neighbors, when a node identifies a cluster head, and waits for its confirmation. We are more likely to encounter such situations at the beginning of the algorithm. Moreover as very few nodes are clustered in the initial phase, more clusters are created, leading to more exchanged messages and measurements.

We can observe from figure 8.10(a) that, as expected, the cumulative number of messages sent by the "Selfish variant" is less important than the cumulative number of messages sent by the "Cooperative variant". It is however important to observe that the number of exchanged messages induced by newly created clusters is very low after the initial phase. Once the system reaches the steady state, no more messages, are exchanged. The low message exchange rate observed during the initial phase is confirmed by our results in table 8.1 with very low message rates of the scale of 10^{-4} per node per second as average values, and a maximum of 0.635 message/node$\times$second.



(a) Cumulative number of exchanged messages (b) Cumulative number of performed measurements

Figure 8.10: Induced overhead: exchanged messages and performed measurements.

In figure 8.10(b) we observe the cumulative number of measurements performed towards the identified cluster heads (typically ping messages). We see again that the major overhead is induced during the initial phase. We observe more pings initially, when nodes initiate their clustering process, with a maximum ping rate of 0.034 ping/node$\times$second. Such very low overall ping rate per node is promising for large scale deployment of our clustering scheme.

8.2.4 Conclusion

In this section, we have shown that Vivaldi's estimations can be exploited to construct efficient clustering schemes in a distributed way. Indeed, the proposed clustering process is based on a first approximation of node positioning using coordinates, along with an adaptive back-off strategy that allows load-balanced construction of clusters. We presented two variants of such clustering process that trade off the convergence time against the induced overhead. However, the two variants have been shown to be effective, enjoying good clustering performance, while achieving a good trade-off between scalability and convergence time. Indeed, nodes are able to identify and join their clusters in a reasonable amount of time and they still do not trigger too frequent measurements.

The reader should note though, that we do not yet consider churn situations, when nodes join and leave the system running the Internet Coordinate System in an asynchronous way. If we consider highly-dynamic networks, the differentiation between the initial phase and the steady state may fade away. However, our clustering process would adapt to situations when only few nodes are existing in the system, by simply not creating clusters if they are "useless". Basically, by setting a minimum number of nodes per cluster at the level of the QT-clustering, low populated clusters could be avoided. Finally, even if we do not address the problem of clusters maintenance, we note that different solutions to such issues have been proposed elsewhere (*e.g.* [45, 21]).

8.3 Shortcut detection using the Two-Tier Vivaldi

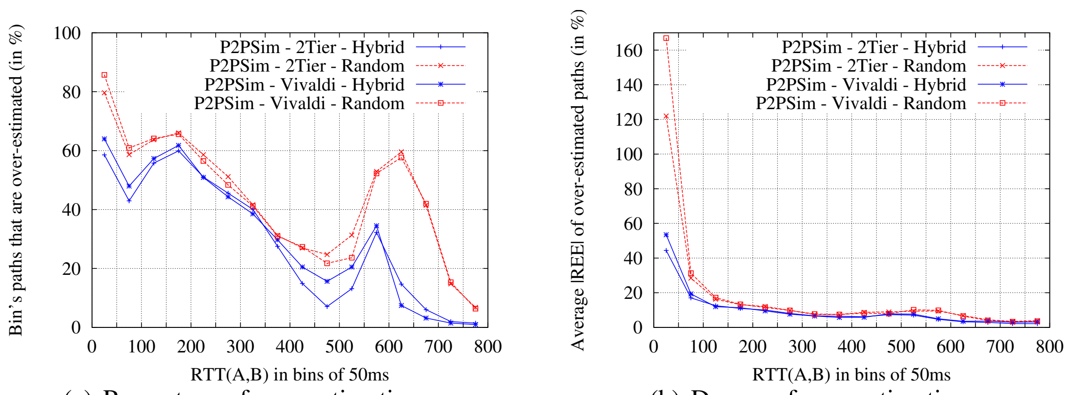
In section 8.1, we have seen that the hierarchical version of Vivaldi improves the quality of the estimations compared to a classical flat Vivaldi. In the current section, we will evaluate if the Two-Tier Vivaldi estimations are more suitable than flat Vivaldi estimations for our one-hop routing shortcuts detection criteria. The self-organized clustering scheme presented in section 8.2 could build clusters for the Two-Tier Vivaldi but, in the current section, we will work with the clusters defined in figure 8.2(a). We have made this choice mainly for simplicity. Indeed, the clusters obtained with the technique presented in section 8.2 require a version of Two-Tier Vivaldi where each node can manage multiple local coordinates (one for each cluster it belongs to). Even if the Two-Tier Vivaldi implementation can be easily adapted to work with multiple local coordinates, the fact that nodes can have multiple coordinates introduces new problems. For example, if two nodes A and B both belong to a cluster c_1 and a cluster c_2, we must decide which local coordinates (those computed in c_1 or those computed in c_2) will be used to estimate the path AB. Since our goal is just to evaluate if the Two-Tier Vivaldi estimations can improve the detection results compared to what we obtained using Vivaldi estimations, we want to avoid such problems and we forbid overlapping clusters. We can obtain such constraint by choosing $k = 1$ in the parameters of the self-organized clustering technique. However, the clusters obtained with such constraint are similar to those defined in figure 8.2(a). Thus, it is simpler to directly use the clusters defined in figure 8.2(a).

8.3.1 Characteristics of the estimation errors

We begin with an analysis of the characteristics of the estimation errors in order to know if the Two-Tier Vivaldi is a suitable estimation mechanism for our criteria. Following the discussion in section 5.9, remember that EDC requires essentially an estimation mechanism that minimizes the over-estimation of the small paths. For ADC and HDC this characteristic is of minor importance. The most important feature for these criteria is to work with a hybrid neighbors selection scheme.

To check the quality of the estimation of the small paths, we divide the whole range of RTTs in the P2PSim data set into equal bins of 50 ms each. For each bin, figure 8.11 gives a description of the over-estimations experienced by the bin's paths: figure 8.11(a) shows the percentage of bin's paths that are over-estimated and figure 8.11(b) shows the average of the $|REE|$ observed for bin's paths that are over-estimated.

In figure 8.11(a), we see that, with Two-Tier Vivaldi applied with hybrid neighbors (resp. randomly selected neighbors), less small paths are over-estimated than with Vivaldi applied with hybrid neighbors (resp. randomly selected neighbors). However, Two-Tier Vivaldi applied with randomly selected neighbors over-estimates more small paths than Vivaldi applied with hybrid neighbors. In figure 8.11(b), we see that the over-estimated small paths are less over-estimated with Two-Tier Vivaldi applied with hybrid neighbors (resp. randomly selected neighbors) than with Vivaldi applied with hybrid neighbors (resp. randomly selected neighbors).



(a) Percentage of over-estimations (b) Degree of over-estimation

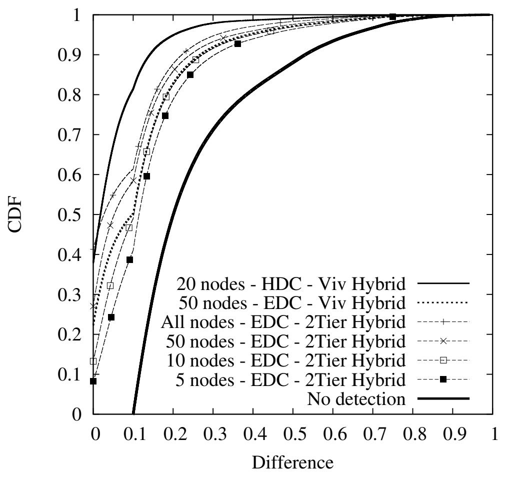
Figure 8.11: Over-estimations of the paths obtained with the Two-Tier Vivaldi.

In the light of these observations, we conclude that using the Two-Tier Vivaldi estimations (computed with hybrid neighbors) should give better detection results than using Vivaldi estimations (computed with hybrid neighbors). We can also say that Two-Tier Vivaldi estimations computed with randomly selected neighbors should not give good detection results. These conclusions are at least valid for EDC. For ADC and HDC, it is clear that Two-Tier Vivaldi applied with randomly selected neighbors is not suitable due to its neighbors selection scheme. On the point of the neighbors selection scheme, Two-Tier Vivaldi applied with hybrid neighbors and Vivaldi applied with hybrid neighbors are tied. However, we should have a small improvement of the detection results when using Two-Tier Vivaldi estimations since the quality of the estimations of the small paths is also important.

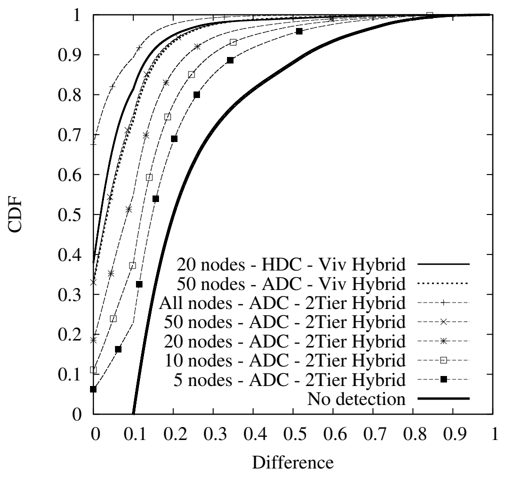
8.3.2 Shortcut detection results

Since it is clear that Two-Tier Vivaldi applied with randomly selected neighbors is not suitable for our shortcuts detection criteria, we will focus here on Two-Tier Vivaldi applied with hybrid neighbors. The detection results obtained with that estimation mechanism are illustrated in figure 8.12. As usually, the quality of the detection results is expressed as the potential improvement remaining after the detection process for each path. In other words, for each path (for which there exists at least one interesting shortcut), we compute the difference between the gain provided by the best existing shortcut and the gain provided by the best detected shortcut. The curves in figure 8.12 represent the CDFs of these differences. Remember that the faster such CDF rises, the better it is because it indicates that lots of paths have a small potential improvement remaining after the detection process (*i.e.* we were able to detect the best shortcut or, at least, one of the best shortcuts for lots of paths).

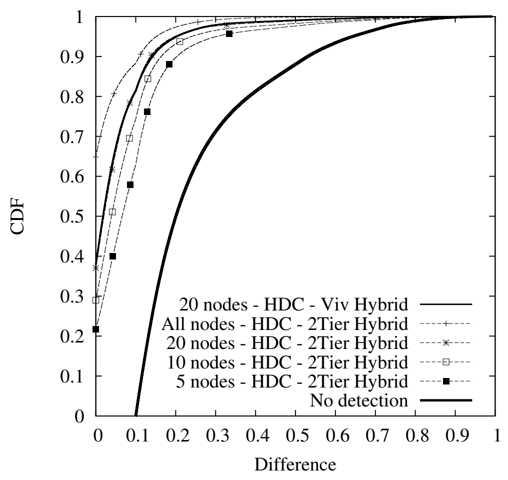
In figure 8.12(a), we see that EDC produces better detection results with Two-Tier Vivaldi than with Vivaldi. The first observation is not encouraging because EDC applied with Two-Tier Vivaldi and $k = 50$ does not produce significantly better results than EDC



(a) EDC



(b) ADC



(c) HDC

Figure 8.12: Difference of G_r between the best existing shortcut and the best detected shortcut when the criteria are applied with estimations produced by a Two-Tier Vivaldi. These results have been computed using the P2PSim data set.

applied Vivaldi and $k = 50$: the remaining potential improvement is smaller than 0.2 for 86% of the paths with the first one while it is smaller than that value for 82% of the paths with the second one. However, we can also see that EDC applied with Two-Tier Vivaldi and $k = 10$ produces almost the same detection results (in terms of quality) than EDC applied with Vivaldi and $k = 50$. Thus, to reach the same level of quality, working with Two-Tier Vivaldi requires to check with measurements five times less candidates than when we work with Vivaldi. This is a significant improvement but HDC applied with Vivaldi and $k = 20$ still provides better results than what we can obtain with EDC applied with Two-Tier Vivaldi. Thus, HDC applied with Vivaldi and $k = 20$ remains our reference point.

With ADC and HDC, we see respectively in figures 8.12(b) and 8.12(c) that the improvement obtained by replacing Vivaldi by Two-Tier Vivaldi is really small. Generally, when Vivaldi is replaced by Two-Tier Vivaldi, there a small improvement in terms of remaining potential improvement but it is smaller than 1%. This is due to fact that both estimation mechanisms use a suitable neighbors selection scheme for the criteria and produce quite accurate estimations allowing to select the right node when ADC searches C's nearest node among A's (resp. B's) neighbors. For this search, accurate estimations for the small paths may help. However, since Two-Tier Vivaldi (in the form used here) only improves a small part of the paths (less than 12% of the paths), the impact of the estimations accuracy improvement on the detection results obtained with ADC and HDC appears finally negligible .

In conclusion, HDC applied with $k = 20$ is still our best detection criterion. Following the results presented in figure 8.12(c), HDC can be applied either with Vivaldi, or with Two-Tier Vivaldi. However, if more paths can be estimated as intra-cluster paths[2] using the Two-Tier Vivaldi, this estimation mechanism can potentially improve the estimation accuracy of a large amount of paths compared to Vivaldi. In such conditions, the quality of the detection results obtained using the Two-Tier Vivaldi estimations should be significantly better than the quality of the detection results obtained using Vivaldi estimations.

8.4 Conclusion

After two failed attempts[3] at replacing Vivaldi by another estimation mechanism to improve the shortcut detection results we propose Two-Tier Vivaldi. This estimation mechanism defines clusters of near nodes and runs a local Vivaldi system in each cluster. The coordinates computed by such local Vivaldi can only be used to estimate intra-cluster paths but, since the clusters have a limited diameter, the clusters only contain a small number of TIVs and intra-cluster path estimations are more accurate than the estimations obtained when all the nodes participate in the same Vivaldi. The local Vivaldi instances form the lower layer of Two-Tier Vivaldi while the upper layer is composed of a Vivaldi where all the nodes participate (just like a classical Vivaldi) in order to be able to estimate inter-cluster paths. Thus, Two-Tier Vivaldi is able to improve the accuracy of intra-cluster paths (*i.e.* small paths) estimations without degrading the accuracy of long paths estimations. Moreover, thanks to a particular neighbors selection scheme, Two-Tier Vivaldi does not consume significantly more resources than Vivaldi: the only difference is that the nodes have to exchange two coordinates (the local one and the global one) and the ID of the cluster they belong to, instead of just one coordinate.

For the shortcut detection problem, replacing Vivaldi estimations by Two-Tier Vivaldi estimations improves the detection results with EDC and slightly improves the detection results with ADC and HDC. However, our tests have been performed with only three

[2]Such goal can be reached by defining more clusters than the three clusters we used for the results we presented here. Following the observations presented in section 8.1, these three clusters are the only natural major clusters obtained for the Vivaldi data set. Thus, estimating more paths as intra-cluster paths seems difficult without considering overlapping clusters. To obtain such clusters, we can use the clustering procedure presented in section 8.2.

[3]Non-linear transformations failed at improving the quality of the estimations and DMF improved the accuracy of the estimations but did not work with the suitable neighbors selection scheme.

clusters and the intra-cluster paths represent only 12% of the total number of paths. Thus, if we consider the whole network, Two-Tier Vivaldi only improves the accuracy for a small percentage of the total number of estimations and the impact of the improvement on the detection results is obviously limited (particularly with ADC and HDC for which the accuracy of the small paths estimations in not the most crucial point). However, defining more clusters and authorizing the clusters to overlap may increase the percentage of paths that can be estimated as intra-cluster paths and, thus, increase Two-Tier Vivaldi's positive impact.

All the results presented in this chapter have been obtained using manually built clusters. In order to keep the self-organized and distributed Vivaldi's important characteristics we proposed a self-organized clustering scheme. We proposed two variants of the algorithm and both variants have been shown to be effective, enjoying good clustering performance, while achieving a good trade-off between scalability and convergence time.

Finally, even with Two-Tier Vivaldi, HDC applied with $k = 20$ remains our best shortcuts detection criterion. With that criterion, the results obtained with a basic implementation of Two-Tier Vivaldi (without overlapping clusters) are similar to those obtained with a classical Vivaldi.

Chapter 9

Future Work

Abstract

In this chapter, we will present some possibilities to improve or extend the work presented in this thesis. Obviously, the main improvement for our estimation-based approach is to find a way to reduce its communication cost. This problem will be discussed in section 9.1. Then, in sections 9.2 and 9.3 we will discuss extensions of our work. these extensions are respectively considering other metrics than delays and considering multihop shortcuts. Finally, in section 9.4 we will discuss the possibility to replace RON's peer-to-peer approach (overlay nodes are end-hosts) by an approach where nodes can also be routers.

9.1 Reducing the communication costs

In its current state, the main drawback of our approach is the communication cost. As discussed previously, compared to a measurement-based approach, we reduce drastically the measurement costs ($O(n \times m)$ with $m \lll n$ instead of $O(n^2)$) but our communication costs are still important. Indeed, even if they are $O(n^2)$ instead of $O(n^3)$, $O(n^2)$ is not really scalable. Finding a way to reduce these communication costs would be a big advantage for our estimation-based approach. During this thesis, we have not extensively studied this problem but we will discuss some elements in the current section.

9.1.1 Ensuring the knowledge of every coordinate for every node

In the current version of our routing shortcut detection mechanism, we consider that each node A must know the coordinate of every node in the overlay in order to be able to estimate AC and CB for any destination B and any potential shortcut C. To obtain such result, after an update of its coordinate, each node sends its new coordinate to every nodes in the overlay. It represents a permanent $O(n)$ communication traffic per node. Thus, we have a total communication cost that is $O(n^2)$.

If we want to keep the requirement stating that a node must permanently have a local copy of the current coordinate of every overlay node, it seems difficult to do better than

$O(n^2)$. Indeed, if each node must permanently get the coordinates of $n-1$ other nodes, then the incoming traffic at each node will obviously be $O(n)$ and the total communication cost will be $O(n^2)$. Thus, we will probably have to deal with a partial knowledge of the coordinates.

9.1.2 Partial knowledge

Partial knowledge of the coordinates is not really a problem for our shortcut detection mechanism. If a node A must search a shortcut between itself and a given destination B, it just requires its own coordinate and B's coordinate. If the coordinate of a given node C is unknown, this node will simply not be considered as a potential shortcut. Thus, the only consequence of a partial knowledge of the coordinates is that we may miss some shortcuts. In fact, the most important is to ensure the knowledge of the coordinates of the nodes that are potential shortcuts. For instance, a node A does not need the coordinate of a node C that is far from him since this node will probably never be the most interesting shortcut for any given path AB. Consequently, we can consider solutions like sending the coordinate to the nearest neighbors of the node with these neighbors relaying the coordinate to their own nearest neighbors, etc. Such solution can provide enough information to provide good detection results with a communication cost smaller than the basic "everybody sends its coordinate to everybody". Since we have not studied the impact of a partial knowledge of the coordinates during this thesis, this remains hypothetical and should be investigated as a future work.

9.1.3 Partial knowledge with the advantages of a full knowledge

In sections 9.1.1 and 9.1.2 we considered that the search of a shortcut for a given path AB is done by the source of that path, *i.e.* A. Since that node has to do the search alone, it has to know the coordinate of every[1] node in the network in order to be able to decide, for each node, if it is an interesting relay or not. But, the shortcut search for a given path AB can be distributed among multiple computing nodes. If each C node is known by at least one of these computing nodes, then computing nodes can deal with partial knowledge of the coordinates without missing potential shortcuts.

Sontag *et al.* [77] proposed to do something similar to reduce RON's communication cost. Their approach allows each node to find an optimal one-hop path to any other node with a communication cost $O(n^2 \times \sqrt{n})$ instead of $O(n^3)$. To distribute the measurement results among nodes, Sontag *et al.* propose to put the n nodes in a grid of size $\sqrt{n} \times \sqrt{n}$ as indicated in figure 9.1. For the node in position (i, j), we define its *rendezvous node set* as the set of all the nodes in row i and in column j. For instance, in figure 9.1, the rendezvous node set for node 1 (resp. for node 9) is $\{2, 3, 4, 7\}$ (resp. $\{3, 6, 7, 8\}$). This way, every pair of node shares at least one rendezvous node. If every node sends its measurements results to its rendezvous nodes, for every path AB, there exists at least one rendezvous node that has the results of the measurements performed by A and the results of the measurements performed by B. This rendezvous node is able to find a shortcut for

[1]Or, at least, it has to know the coordinate of as many nodes as possible.

the path AB. For instance, in figure 9.1, node 3 and node 7 are both able to find a shortcut for the path between node 1 and node 9. Thus, if every node sends its measurement results to its rendezvous nodes (the total number of messages is $O(n \times \sqrt{n})$ with a $O(n)$ message size) and if a node sends a request to all its rendezvous nodes when it wants to find a shortcut towards a given destination, the system is able to find the best shortcut for any given path AB.

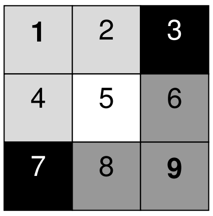


Figure 9.1: Communication cost reduction with the approach proposed in [77]. Example of grid for 9 nodes.

We can do something similar with an estimation-based approach. Indeed, if each node sends its coordinate to its rendezvous nodes, we generate a communication traffic that is $O(n \times \sqrt{n})$. Since, for a given path AB, the coordinate of each C node is known by at least one A's rendezvous node, we can imagine the following mechanism. When a node A wants a shortcut towards a destination B, A measures $RTT(A, B)$ (as indicated in line 1 of algorithm 3 page 87) and gets B's coordinate simultaneously. Then, it gives these information to its rendezvous nodes and it asks them their k best candidates (among the C nodes they know) for the path AB. Thus, A receives k candidates from each rendezvous node and it can rank these candidates with respect to the estimated gain they provide. Node A keeps the k best candidates and checks them through measurements in order to find the best shortcut among them. This way, the result should be exactly the same as the result obtained using algorithm 3 but our communication costs are $O(n \times \sqrt{n})$ instead of $O(n^2)$. Testing such an approach is also a future work.

9.2 Considering more metrics than the RTT

In this thesis we focused only on the RTT as path performance indicator. Thus, we are able to find routing shortcuts with respect to RTTs but not with respect to other metrics. Considering only RTTs is sufficient for lots of applications (*e.g.* Voice over IP, online video games, *etc.*) but, for other applications, the RTT is an important metric but not the only one. For instance, a video-conference application requires simultaneously small delays and large bandwidth. Thus, for such applications, just focusing on RTTs to describe the performance of a given path is not sufficient.

At the beginning of the thesis, coordinate systems were limited to delay estimations and estimating other metrics was always announced as a future work. Thus, limiting our work to RTTs was a natural choice. Today, things are different and some estimation mechanisms like DMF [41] and Sequoia [71] have been recently proposed to provide bandwidth estimations. Analysing the characteristics of these estimations[2] in order to find a way to use them to detect routing shortcuts with respect to other metrics is a possible extension of the work presented in this thesis.

However, before doing this analysis, the first thing to do is to establish if searching shortcuts with respect to a given metric is relevant or not. Indeed, if we consider the RTTs, the performance of a given path is the sum of the delays experienced on that path. Thus, modifying the path followed by the packets inside the network changes the result of the sum and, finding a path providing better performance than the default Internet path is possible. Consider now the bandwidth as performance indicator. The performance of a given path is the minimum available bandwith experienced on links used by that path. But, nowadays, bandwidth bottlenecks are generally customers' Internet access link. In such situation, using a relay node to modify the Internet path does not improve the performance. Indeed, even if the path in the Internet is modified, the customer still uses its Internet access link and the minimum available bandwith remains the same. In conclusion, searching routing shortcuts with respect to any given metric will not always be relevant.

9.3 Extending to multihop routing shortcuts

We have already said in chapter 2 that previous studies (*e.g.* [51]) concluded that limiting alternative routes to one intermediate hop is sufficient to solve many performance failures. Nevertheless, considering multihop shortcuts rather than only one-hop shortcuts can obviously improve the gains provided by the shortcuts. To quantify how big this improvement is, for each path, we can compare the relative gain provided by the best shortcut, namely Gr_{best}, and the gain provided by the best one-hop shortcut, namely $Gr_{best1hop}$. Considering every path for which there exists at least one shortcut, figure 9.2 gives the CDF of the differences between these two values.

In figure 9.2, we see that considering multihop shortcuts provides better shortcuts than when we consider only-hop shortcuts because the difference is not equal to zero for most of the paths. However, in most cases, the improvement is not huge. For instance, the difference is bigger than 20%, for about 40% of the paths in P2PSim, about 30% of the paths in Meridian and almost no path in Planetlab. Thus, for most of the paths, just considering one-hop shortcuts can provide already a good delay improvement and considering multihop shortcuts only allows us to improve slightly more the delays. Moreover, the delays computed to build the curves presented in figure 9.2 just take into account the RTTs of the paths composing the alternative paths. We did not consider the processing times introduced by the relays. If a multihop alternative path uses, for instance, 5 relays to reach the destination, the processing times would become important compared to a one-hop shortcut and many multihop "shortcuts" can finally be detours.

[2]With delays, we have seen that simply replacing the measurements by Vivaldi estimations does not work. We based our detection criteria on an analysis of the characteristics of the RTT estimations. There could be similar problems with other metrics, and an analysis of their estimations should be necessary to be able to find good shortcuts.

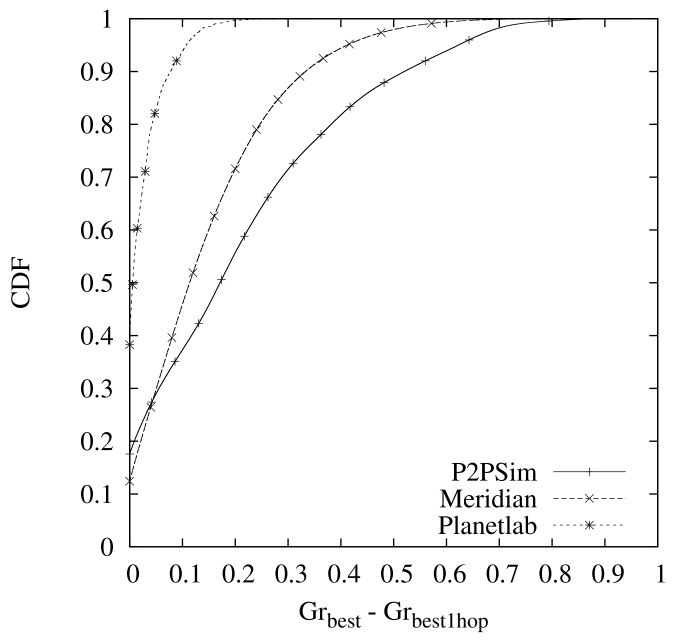


Figure 9.2: Difference between the relative gain provided by the best shortcut and the relative gain provided by the best one-hop shortcut for every path for which there exists at least one shortcut.

The reader should also notice that extending the shortcut search to multihop shortcuts also increases the complexity of the shortcut search. Indeed, if we focus only on one-hop shortcuts, when A wants to find a shortcut towards a given destination B, it has only to consider a number of alternative paths which is $O(n)$ (where n is the number of nodes in the network). Considering multihop shortcuts will increase the computation costs and the time before being able to provide an answer to the application that asked for a shortcut. Since the improvement in quite small for most of the paths, the disavantages perceived by the users can finally be bigger than the advantages.

9.4 Bringing the shortcut finder inside the network

In this thesis, we investigated the feasibility of an estimation-based overlay routing mechanism. We proposed some shortcut detection criteria using estimations as inputs and we showed that these criteria can improve the delays for lots of paths in the Internet. These criteria use the concept of overlay node: the nodes share information and a node is able to decide if another node is a shortcut towards a given destination. In a basic view of the problem, overlay nodes are end-hosts because deploying a new service on such nodes is easy (the agreement of the ISPs is not required) and justifiable (the users generally have benefits in taking part in the overlay network). Basically, our shortcut detection mechanism is a peer-to-peer mechanism: it has been designed to be deployed on end-hosts between the applications and the TCP/IP stack.

Even if considering that overlay nodes are only end-hosts is the easiest solution, other approaches can provide better results. For instance, figure 9.3 shows that using routers as overlay nodes should allow better shortcut findings. Indeed, in this example, the alternative path ACB crosses two times the link between C' and C: one time to go from C' to the relay C and, the other time, to come back from the relay to the network. In such situation, using C' as relay provides a better alternative path than using C as relay. However, C' is not an end-host and we must find a way to bring our shortcut detection mechanism inside the network instead of doing all the operations at the network edge.

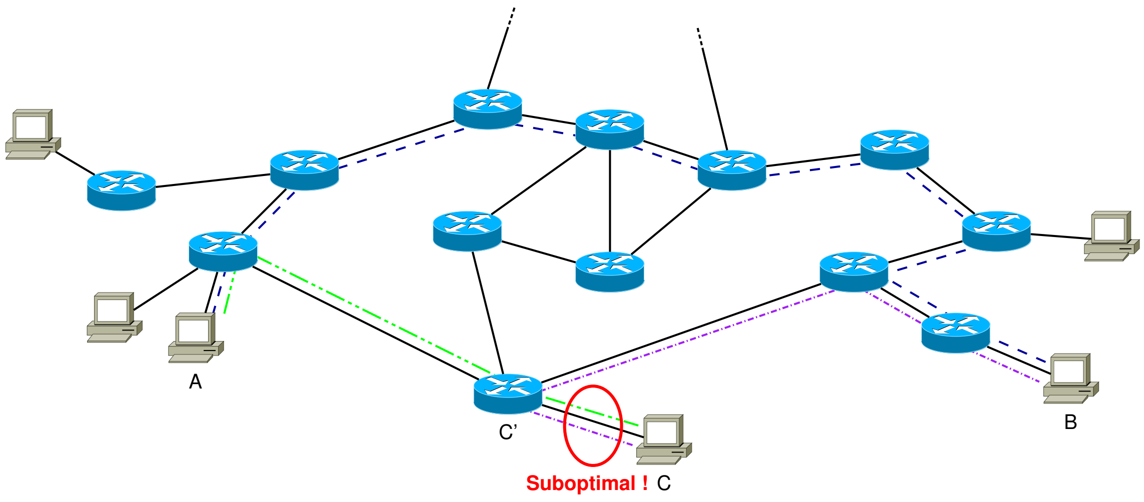


Figure 9.3: **Suboptimality of the P2P approach.** With a P2P approach a relay C is an end-host. Thus, C's Internet access link is crossed two times (once in each direction). If the router C' is an overlay node the alternative path $AC'B$ provides a smaller delay than the path ACB.

Since our detection criteria can be directly applied even if overlay nodes are not end-hosts, the question is not to decide how we detect shortcuts. The problem is to know how the system would work. With the P2P approach, answering this question was easy: if A wants to send a packet to a given destination B, A searches a shortcut towards that destination and A manages the indirect sending to the destination B through the relay C. When nodes can also be routers, answering this question is more or less difficult depending of the approach used.

The most simple approach to bring the shortcut detection mechanism inside the network is to consider routers as normal overlay nodes: they compute their coordinates and they share their coordinates with the other nodes. With such an approach there is no difference compared to the P2P approach. Each node continues to perform its own shortcut searches and it will simply have routers among the C nodes to consider as potential shortcuts. The major drawback of this approach is the lack of incentives for the ISPs to take part in the overlay. Indeed, even if Lumezanu *et al.* [49] have shown that there are many cases where shortcuts do not violate routing policies, the ISPs have no control on the shortcuts actually used with this approach. Moreover, this approach adds some computation tasks to the routers (computing the coordinate, sharing the coordinate,...) and these additional tasks can have a bad impact on router performance.

To avoid these drawbacks, another approach is to deploy a dedicated infrastructure for the computation of router coordinates (to reduce their computation load) and to let this infrastrucure manage the shortcut searches (to allow the ISPs to keep the control on the shortcuts used). Basically, each ISP can deploy a *Coordinate Server* (CS in short) and this CS computes the coordinates of the ISP routers. Thus, when they receive a request from the CS, the routers just have to perform measurements towards their neighbors and to return the measurement results to the CS. To allow ISPs to select the best shortcuts without violating the routing policies, CSes are also responsible for the shortcut searches. Thus, the end-hosts are not responsible for the shortcut searches any longer and the shortcut search process becomes more complicated. A possible shortcut search process is described in figure 9.4. When an ISP edge-router A receives a packet that must be routed using the shortest path with respect to delays, it requests a shortcut for that packet to its CS. If the CS finds a shortcut, it returns the information to A. Then, A sends the packet to the shortcut and the shortcut forwards the packet to its destination.

Even if the shortcut search process described in figure 9.4 seems feasible and more efficient than the P2P approach, there are some problems for which we have currently no solution (or only simple and inefficient solutions). Answering the following questions (and others that will probably appear with further investigations) is a future work:

How can A discover if a given packet must be routed through a shortcut? A first solution is to add a flag in the packet (TOS byte, DSCP byte) to indicate if the packet is delay sensitive or not. This flag can be activated by the application generating the packet (thus, by A_1 in figure 9.4). To have a complete transparency for the end-hosts, another solution is to analyse the traffic at the edge-routers to differentiate delay sensitive traffic from normal traffic.

For which path does CS_A search a shortcut? When A requests a shortcut, it is for the path between itself and B_1. But B_1 is not an overlay node and has no coordinate. In figure 9.4, we consider that CS_B, *i.e.* the CS of B_1's ISP, is able to propose one of its overlay node to represent B_1. However, having only information about B_1, we do not actually know how to find node B.

What happens if A_1's (resp. B_1's) ISP does not participate in the overlay? The most efficient solution is to allow A_1 (resp. B_1) to take part in the overlay using the P2P approach: A_1 (resp. B_1) will compute its coordinate and will search its own shortcuts. From CS viewpoints, A_1 (resp. B_1) will appear as being simultaneously a CS and an overlay node but it should not be a problem.

What about node addresses? With a P2P approach, nodes were end-hosts. Thus, each node had generally only one IP address and there was no problem. If nodes are routers, things are different. Indeed, a router has multiple addresses and each address can be used to send data to the router. As shown in figure 9.5(a), depending of the address used to reach a router, the path followed can be different. Thus, measurement results can be different and conclusions about shortcuts can be different. For instance, in figure 9.5(a), if R_1 uses $1.1.3.1$ to reach R_2 and, if R_1's routing table indicates that R_3 is the next hop for $1.1.3.0/24$, then measurements between $R1$ and $R2$ are measurements of the path $R_1R_3R_2$ and R_2 can never be perceived

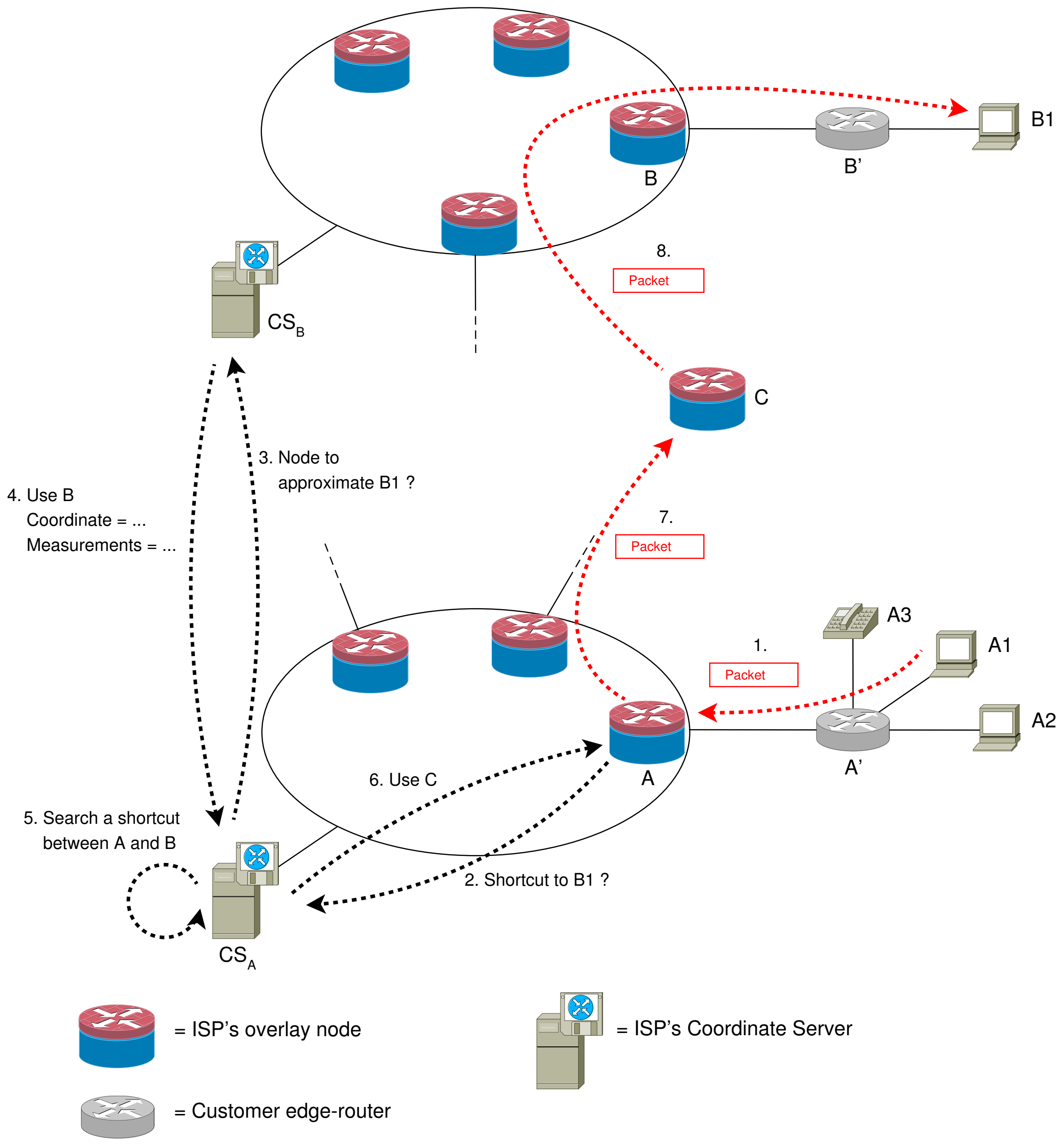
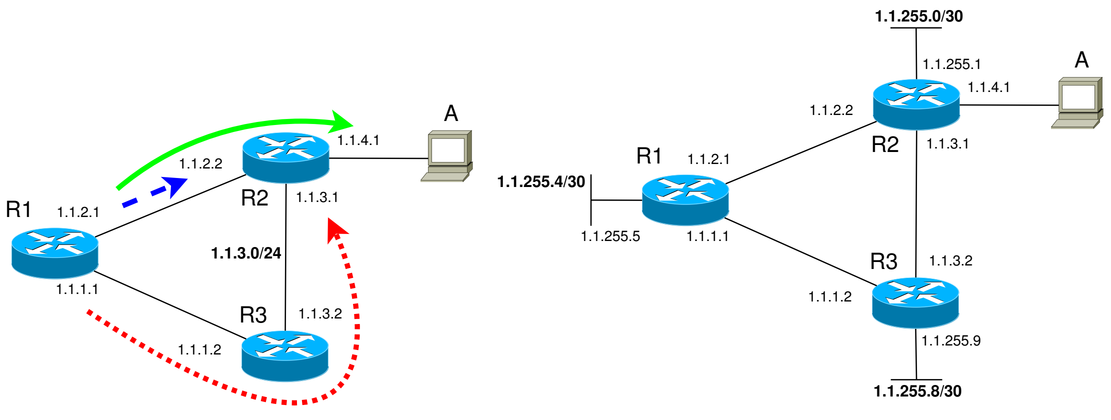


Figure 9.4: **Possible shortcut search process with nodes inside the network.** When an ISP's edge-router receives a packet that must be routed through a shortcut, it sends a request to its CS to obtain a shortcut for the packet's destination, and it is responsible for the management of the indirect routing through the shortcut.

as a shortcut for the path R_1R_3. On the other hand, if R_1 uses $1.1.2.2$ to reach R_2, then measurements between R_1 and R_2 are measurements of the direct path between these nodes and R_2 can potentially be considered as a shortcut for the path R_1R_3. To avoid such problem, a simple solution is proposed in figure 9.5(b). It simply consists in attaching a virtual $/30$ network[3] to every router and to address each router using its virtual interface. For instance, in figure 9.5(b), we have added an interface to R_2 with $1.1.255.1$ as address. Since $1.1.255.0/30$ can only be reached through R_2 we are sure that the packets will cross R_2 to reach that destination. Thus, from any source in the network, packets will follow the shortest path to R_2 to reach $1.1.255.1$ and our problem is solved. However, that solution "uselessly" consumes IP addresses and it should be possible to find a more convenient way to solve that problem.



(a) Problem (b) Simple Solution

Figure 9.5: **Problems with nodes' addresses.** Figure 9.5(a) shows that, depending of the address used, different paths to reach a same node can be used. In order to solve the problem, figure 9.5(b) proposes to add virtual interfaces to the nodes and to use these interfaces to address packets to the nodes.

9.5 Conclusion

In this chapter, we investigated four big potential improvements for the work presented in this thesis: finding a way to reduce the communication costs, considering other metrics than the RTTs, extending the shortcuts search to multi-hops shortcuts and, finally, replacing the classical peer-to-peer approach by an intra-network approach. These are, in our mind, the four most important improvements/extensions for our work. But that list is not exhaustive and some other elements can be investigated. For instance, we should consider the concept of mutual advantage proposed by Lumezanu *et al.* [51, 48] in order to improve the incentive for the nodes to take part in the overlay.

[3]If it does not generate routing problems, we can even use $/31$ virtual networks for this purpose.

Finally, future work can also include detection tests with other detection criteria and/or other estimation mechanisms. In this thesis, we performed tests with some detection criteria (EDC, ADC, HDC, SDC, *etc.*) and some estimation mechanisms (Vivaldi, DMF and Two-Tier Vivaldi, each time with different neighbors selection schemes). Our best detection results have been obtained using Vivaldi (or Two-Tier Vivaldi) with HDC and $k = 20$. These results allow us to conclude that finding shortcuts using an estimation-based detection mechanism is feasible but these results can potentially be improved by using other detection criteria and/or estimation mechanisms.

Chapter 10

Conclusion

In this thesis, our main goal was to investigate the feasibility of an estimation-based approach of overlay routing to replace RON's unscalable measurement-based approach. We showed that, for any given path AB, using only the RTT of that path and the information available in an ICS (namely, Vivaldi), it is possible to select a small set of nodes containing very likely an interesting one-hop shortcut (but not necessarily the best one) when shortcuts exist for that path. We obtained the best results with our shortcut detection criterion called HDC. With that criterion we are able to limit the number of potential shortcuts for any given path AB to about one or two percents of the total number of nodes in the network. So, to improve significantly the latency between A and B, we will only have to perform measurements between A, B and these few candidate nodes to know if they are really shortcuts and which of them is the best shortcut. These results are encouraging and it should be possible to obtain better detection results with a fine tuning of the parameters or even with other estimation-based detection criteria and/or other estimation mechanisms.

The estimation-based approach proposed in this thesis is significantly more scalable than RON's measurement-based approach. RON's solution has a global measurement cost which is $O(n^2)$ and a global communication cost which is $O(n^3)$ where n is the number of nodes participating to the overlay. With our estimation-based approach, these costs become respectively $O(n \times m)$ and $O(n^2)$ where m is the number of neighbors used by each node to compute its Vivaldi coordinate ($m \lll n$). Even if our estimation-based approach is more scalable than a measurement-based approach, its communication cost can still be problematic at very large scale. Even if this problem has not been deeply investigated in the thesis, we have already proposed some solutions to reduce the communication cost. For instance, with a solution similar to what is proposed by Sontag *et al.* in [77], it should be possible to reduce the communication cost to $O(n \times \sqrt{n})$ without any loss of accuracy for the shorcut detection. Nevertheless, finding and testing a solution to reduce the communication cost is still required in order to obtain a routing shortcut detection mechanism which is really scalable.

Compared to a measurement-based approach, we are aware that our estimation-based approach has several shortcomings. The main problem is that we have no guarantee to find the best shortcut for any given path in the network. However, our tests have shown that, even if we cannot always detect the best shortcut, we are able to significantly reduce the RTT for most of the paths AB. The second problem is that we have less reactivity to delay changes than with a measurement-based approach. Indeed, with an estimation-based approach, when delays change, it is necessary to wait a small time period before the estimations reflect these new delays. However, such small problems seem to be the price to pay to obtain a scalable overlay routing mechanism because none of RON's improvements proposed over years is able to reduce the costs of overlay routing as our estimation-based approach does.

Trying to detect routing shortcuts with an estimation-based mechanism and doing experiments with different estimation mechanisms and different detection criteria was the main part of our work but not the only one. In the first part of our work we extensively studied shortcuts/TIVs in the Internet and their impact on Vivaldi. The first observation was that larger triangles are more likely (severe) TIVs, with respect to the distribution of RTTs in the Internet. In the light of this observation, we proposed a Two-Tier architecture to mitigate the impact of TIVs on the estimations. This architecture is based on a clustering of the nodes. Within their cluster, nodes use more accurate local coordinates to predict intra-cluster distances, and keep using global coordinates when predicting longer distances towards nodes belonging to foreign clusters. Using Vivaldi, we have shown that estimations of intra-clusters distances are significantly better with the Two-Tier architecture than with a classical flat architecture. It is worth noticing that although we focused on Vivaldi for measurements and experimentations, the Two-Tier architecture we proposed is independent from the embedding protocol used. Our proposed method would then be general enough to be applied in the context of coordinates computed by Internet coordinate systems other than Vivaldi.

A second observation done during our study of shortcuts/TIVs is that they have a real impact on Vivaldi. We observed that TIV bases are generally underestimated and that they have usually a small REE variance. Following that, we clusterized the REE variance of node pairs using GMMs and ARMA models. We obtained satisfactory results with up to 85% of TIV bases detected, while suspecting non-TIV bases in rare situations. Using machine learning techniques, an even more discriminative variable to establish whether a given path is a TIV base or not could be found. This variable, named OREE, combines information about estimation error and REE variance in order to provide an answer for a given path AB. However, even if these TIV base detection techniques have a very high probability of success, they also require a monitoring of the path AB over a quite long period of time before providing an answer for that path. Thus, it is impossible to use these detection techniques in order to know if it is useful or not to check every potential relay node C for a given path AB (*i.e.* as a first step for a general shortcut detection process). Nevertheless, such TIV base detection techniques can be useful in specific situations as, for instance, managing Vivaldi's neighbors in order to avoid the use of TIV bases as neighboring links.

The main part of the results presented in this thesis were based on Vivaldi. Due to its properties, this estimation mechanism seemed to be the best choice at the beginning of our work. However, some solutions to allow estimation mechanisms to deal with TIVs have been proposed after we had made this initial choice. We considered two of these solutions. The first one was proposed to allow Vivaldi to provide accurate estimations even in case of shortcut/TIV. The idea was to apply non-linear transformations to the delays before trying to embed them in the metric space. Unlike what was stated by the designers of this solution, it is not the miracle solution to TIV problems. Indeed, in order to remove all TIVs, we observed that it is necesary to apply very strong non-linear transformations. Even if the transformed delays obtained this way can potentially be accurately estimated, small estimation errors are unavoidable with Vivaldi. Moreover, even if the estimation errors on transformed delays are small, they become very large once the reverse strong non-linear transformation is applied in order to obtain estimations of the original delays. Thus, estimation errors are finally bigger than what we obtain without transformation. Even if non-linear transformations are not the miracle solution to allow Vivaldi to deal with TIVs, we observed that applying small transformations allows, in some cases, to slightly improve the estimation accuracy compared to a classical approach of Vivaldi. We also considered an estimation mechanism, named DMF, that has been developed to naturally deal with TIVs. We observed that DMF is able to provide more accurate estimations than Vivaldi. Nevertheless, we also observed that the DMF estimations have less suitable properties than Vivaldi estimations for our shortcut detection criteria. Thus, DMF did not allow us to improve our shortcut detection results.

In conclusion, the work presented in this thesis is a first step towards a scalable estimation-based one-hop routing shortcut detection mechanism. Even if we showed that detecting shortcuts using estimations instead of measurements is feasible, some work is still required to obtain a scalable solution which is fully operational (essentially finding a way to reduce the communication cost). Working on some potential improvements for the estimation-based approach should also be very rewarding: considering other metrics than delays, considering multihop paths, *etc*. However, even if there is still a lot of work on the subject, stopping our work somewhere is mandatory. The current state of our work seems suitable for this purpose because it already provides an answer to our initial question. Indeed, we are convinced that the estimation-based approach is a promising approach for overlay routing that provides a good trade-off between scalability and quality of the detection results. We hope that the results presented in this thesis have also convinced the reader.

List of abbreviations

ADC	Approximation Detection Criterion
AEE	Absolute Estimation Error
$AEE(X,Y)$	AEE computed for the path XY
ALS	Alternating Least Squares
ARE	Average Relative Error
$ARE(X)$	ARE computed for node X
ARMA	AutoRegressive Moving Average
AS	Autonomous System
BBS	Big-Bang Simulation
BGP	Border Gateway Protocol
CDF	Cumulative Distribution Function
CS	Coordinate Server
DMF	Decentralized Matrix Factorization
DMFSGD	Decentralized Matrix Factorization by Stochastic Gradient Descent
ED	Estimated Delays (matrix)
EDC	Estimation Detection Criterion
EG_a	Estimated Absolute Gain (provided by a TIV)
EG_r	Estimated Relative Gain (provided by a TIV)
EGP	Exterior Gateway Protocol
$ERTT(X,Y)$	Approximation used by the shortcut detector for the path XY
$EST(X,Y)$	Estimation provided by the ICS for the path XY
FPR	False Positive Rate
G_a	Absolute gain (provided by a TIV)
G_r	Relative gain (provided by a TIV)
GC	Global Coordinate
GMM	Gaussian Mixture Model
GNP	Global Network Positioning
HDC	Hybrid Detection Criterion
ICS	Internet Coordinate System
IDES	Internet Distance Estimation Service
IGP	Interior Gateway Protocol

ISP	Internet Service Provider
ITPR	Interesting True Positive Rate
LC	Local Coordinate
MD	Measured Delays (matrix)
NMF	Non-Negative Matrix Factorization
NPS	Network Positioning System
P2P	Peer-to-Peer
PIC	Practical Internet Coordinates
RDC	Random Detection Criterion
OREE	Oscillation and Relative Estimation Error
QoS	Quality of Service
REE	Relative Estimation Error
$REE(X,Y)$	REE computed for the path XY
ROC	Receiver Operating Characteristic
RON	Resilient Overlay Network
RTT	Round Trip Time
$RTT(X,Y)$	RTT measured for the path XY
SDC	Simple Detection Criterion
SGD	Stochastic Gradient Descent
SVD	Singular Value Decomposition
TED	Transformed Estimated Delays (matrix)
TIV	Triangle Inequality Violation
TMD	Transformed Measured Delays (matrix)
TPR	True Positive Rate
VoIP	Voice over IP

Bibliography

[1] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient overlay networks. *SIGOPS Oper. Syst. Rev.*, 35(5):131–145, 2001.

[2] L. Bottou. Online algorithms and stochastic approximations. In *Online Learning and Neural Networks*. Cambridge University Press, 1998.

[3] R. Braden, D. Clark, and S. Shenker. Integrated services in the Internet architecture: an overview. RFC 1633, June 1994. http://tools.ietf.org/html/rfc1633.

[4] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource reservation protocol (RSVP). RFC 2205, September 1997. http://tools.ietf.org/html/rfc2205.

[5] F. Cantin, B. Gueye, M. Kaafar, and G. Leduc. Overlay routing using coordinate systems. In *Proc. of ACM CoNext Student Workshop*, Madrid, Spain, December 2008.

[6] F. Cantin, B. Gueye, M. Kaafar, and G. Leduc. A self-organized clustering scheme for overlay networks. In *Proc. of IWSOS*, LNCS 5343, Vienna, Austria, December 2008.

[7] F. Cantin, B. Gueye, M. Kaafar, G. Leduc, and L. Mathy. Explication et réduction de l'impact des violations d'inégalités triangulaires dans vivaldi (in french). In *Actes de Colloque Francophone sur l'Ingénierie des Protocoles (CFIP)*, Les Arcs, France, March 2008.

[8] F. Cantin and G. Leduc. Finding routing shortcuts using an internet coordinate system. In *Proc. of IWSOS*, LNCS, Karlsruhe, Germany, February 2011.

[9] F. Cantin, G. Leduc, and B. Gueye. Transformation non linéaire des distances : une solution au problème des violations d'inégalités triangulaires dans les systèmes de coordonnées ? (in french). In *Actes de Colloque Francophone sur l'Ingénierie des Protocoles (CFIP)*, Strasbourg, France, October 2009.

[10] Y. Chen, X. Wang, X. Song, E. K. Lua, C. Shi, X. Zhao, B. Deng, and X. Li. Phoenix: Towards an accurate, practical and decentralized network coordinate system. In *Proceedings of the 8th International IFIP Networking Conference*, Aachen, Germany, 2009. Springer-Verlag.

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

[12] Z. Chen, Y. Chen, Y. Zhu, C. Ding, B. Deng, and X. Li. Tarantula: Towards an accurate network coordinate system by handling major portion of TIVs. In *Proc. of the 54th Annual IEEE Global Telecommunications Conference (GLOBECOM'11)*, December 2011.

[13] Y. Chu, S. Rao, S. Seshan, and H. Zhang. Enabling conferencing applications on the internet using an overlay muilticast architecture. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, San Diego, California, USA, 2001. ACM.

[14] M. Costa, M. Castro, R. Rowstron, and P. Key. PIC: Practical Internet Coordinates for distance estimation. In *Proc. of the International Conference on Distributed Computing Systems*, pages 178–187, 2004.

[15] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A decentralized network coordinate system. In *Proc. of SIGCOMM*, Portland, OR, USA, August 2004.

[16] C. de Launois, S. Uhlig, and O. Bonaventure. Scalable route selection for IPv6 multihomed sites. In *Proc. of Networking 2005*, Waterloo, Ontario, Canada, May 2005.

[17] W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[18] B. Donnet, B. Gueye, and M. Kaafar. A survey on network coordinates systems, design, and security. *IEEE Communication Surveys & Tutorials*, 12(4), October 2010.

[19] B. Elser, A. Forschler, and T. Fuhrmann. Tuning Vivaldi: Achieving increased accuracy and stability. In *Proc. of the 4th IFIP International Workshop on Self-Organizing Systems*, IWSOS '09, pages 174–184, Zurich, Switzerland, 2009.

[20] T. Fei, S. Tao, L. Gao, and R. Guerin. How to select a good alternate path in large peer-to-peer systems? In *Proc. of INFOCOM 2006*, pages 1–13, Barcelona, Spain, April 2006.

[21] B. Gueye and G. Leduc. Resolving the noxious effect of churn on Internet Coordinate Systems. In *Proc. of IWSOS*, LNCS, Zurich, Switzerland, December 2009.

[22] K. P. Gummadi, H. V. Madhyastha, S. D. Gribble, H. M. Levy, and D. Wetherall. Improving the reliability of Internet paths with one-hop source routing. In *Proceedings of OSDI*, San Francisco, CA, USA, 2004. USENIX Association.

[23] K. P. Gummadi, S. Saroiu, and S. D. Gribble. King: Estimating latency between arbitrary Internet end hosts. In *ACM Internet Measurement Workshop 2002*, Marseille, France, November 2002.

[24] J. Hawkinson and T. Bates. Guidelines for creation, selection, and registration of an autonomous system (AS). RFC 1930, March 1996. `http://tools.ietf.org/html/rfc1930`.

[25] L. J. Heyer, S. Kruglyak, and S. Yooseph. Exploring expression data: Identification and analysis of coexpressed genes. *Genome Research*, 9(11):1106–1115, November 1999.

[26] J. Jannotti, D. Gifford, K. Johnson, F. Kaashoek, and J. O'Toole. Overcast: reliable multicasting with on overlay network. In *Proceedings of the 4th conference on Symposium on Operating System Design & Implementation*, San Diego, California, USA, 2000. USENIX Association.

[27] I. T. Jolliffe. *Principal Component Analysis*. Springer Series in Statistics. Springer-Verlag, second edition edition, October 2002.

[28] M. Kaafar, F. Cantin, B. Gueye, and G. Leduc. Detecting Triangle Inequality Violations for Internet Coordinate Systems. In *Proc. of Future Networks 2009 workshop*, Dresden, Germany, June 2009.

[29] M. Kaafar, B. Gueye, F. Cantin, G. Leduc, and L. Mathy. Towards a Two-Tier Internet Coordinate System to mitigate the impact of Triangle Inequality Violations. In *Proc. of IFIP Networking 2008*, LNCS 4982, Singapore, May 2008.

[30] R. Keralapura, N. Taft, C. Chuah, and G. Iannaccone. Can ISPs take the heat from overlay networks. In *In HotNets*, San Diego, CA, USA, 2004.

[31] S. Keshav. *An engineering approach to computer networking: ATM networks, the Internet and the telephone network*. Addison-Wesley, 1997.

[32] J. Kurose and K. Ross. *Computer Networking: a top-down approach*. Addison-Wesley, 6th edition, February 2012.

[33] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. Delayed internet routing convergence. In *Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, Stockholm, Sweden, 2000. ACM.

[34] D. Lay. *Linear Algebra and its applications*. Addison-Wesley, 1996.

[35] J. Ledlie, P. Gardner, and M. I. Seltzer. Network coordinates in the wild. In *Proc of NSDI*, Cambridge, UK, April 2007.

[36] D. D. Lee and H. S. Seung. Algorithms for non-negative matrix facorization. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, December 2000.

[37] S. Lee, Z. Zhang, S. Sahu, and D. Saha. On suitability of Euclidean embedding of Internet hosts. *SIGMETRICS*, 34(1):157–168, 2006.

[38] P. Lepropre. Implémentation d'une technique de routage basée sur un système de coordonnées (in french). Master's thesis, University of Liège, 2011.

[39] J. Li, K. Miyashita, T. Kato, and S. Miyazaki. GPS time series modeling by autoregressive moving average method. In *Earth Planets Space*, volume 52, pages 155–162, 2000.

[40] Y. Liao, W. Du, P. Geurts, and G. Leduc. DMFSGD: A decentralized matrix factorization algorithm for network distance prediction. *Submitted to IEEE/ACM Transactions on Networking*, 2011.

[41] Y. Liao, W. Du, P. Geurts, and G. Leduc. Decentralized prediction of end-to-end network performance classes. In *Proc. of CoNEXT 2011*, Tokyo, Japan, December 2011.

[42] Y. Liao, P. Geurts, and G. Leduc. Network distance prediction based on decentralized matrix factorization. In *Proc. of IFIP Networking 2010*, Chennai, India, May 2010.

[43] Y. Liao, M. Kaafar, B. Gueye, F. Cantin, P. Geurts, and G. Leduc. Detecting Triangle Inequality Violations in Internet Coordinate Systems by Supervised Learning - Work in progress. In *Proc. of Networking 2009*, Aachen, Germany, May 2009.

[44] H. Lim, J. Hou, and C. Choi. Constructing Internet Coordinate System based on delay measurement. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement (IMC)*, Miami Beach, FL, USA, October 2003. ACM.

[45] X. Liu, J. Lan, P. Shenoy, and K. Ramaritham. Consistency maintenance in dynamic peer-to-peer overlay networks. *Comput. Netw.*, 50(6):859–876, 2006.

[46] L. Ljung. *System identification toolbox for use with Matlab-users guide*. Natick, Mass., 1995.

[47] E. K. Lua, T. Griffin, M. Pias, H. Zheng, and J. Crowcroft. On the accuracy of embeddings for Internet Coordinate Systems. In *Proc. of the IMC Conference*, pages 1–14, Berkeley, California, USA, 2005. ACM.

[48] C. Lumezanu, R. Baden, D. Levin, N. Spring, and B. Bhattacharjee. Symbiotic relationships in Internet routing overlays. In *Proceedings of NSDI*, pages 467–480, Boston, Massachusetts, USA, 2009. USENIX Association.

[49] C. Lumezanu, R. Baden, N. Spring, and B. Bhattacharjee. Triangle Inequality and Routing Policy Violations in the Internet. In *Proceedings of PAM*, pages 45–54, Seoul, Korea, 2009. Springer-Verlag.

[50] C. Lumezanu, R. Baden, N. Spring, and B. Bhattacharjee. Triangle inequality variations in the Internet. In *Proc. of IMC '09*, pages 177–183, Chicago, Illinois, USA, 2009. ACM.

[51] C. Lumezanu, D. Levin, and N. Spring. Peerwise discovery and negotiation of faster paths. In *In HotNets*, 2007.

[52] C. Lumezanu and N. Spring. Playing Vivaldi in hyperbolic space. In *Proc. of SIGCOMM-IMC*, 2006.

[53] G. Malkin. RIP version 2. RFC 2453, November 1998. http://tools.ietf.org/html/rfc2453.

[54] Y. Mao, L. Saul, and J. Smith. IDES: An Internet Distance Estimation Service for large networks. *IEEE Journal on Selected Areas in Communications*, 24(12):2273–2284, 2006.

[55] G. J. McLachlan and D. Peel. *Finite Mixture Models*, volume 1. Wiley, New York, MA, 2000.

[56] J. Moy. OSPF version 2. RFC 2328, April 1998. http://tools.ietf.org/html/rfc2328.

[57] A. Nakao, L. Peterson, and A. Bavier. A routing underlay for overlay networks. In *Proceedings of SIGCOMM*, pages 11–18, Karlsruhe, Germany, 2003. ACM.

[58] A. Nakao, L. Peterson, and A. Bavier. Scalable routing overlay networks. *SIGOPS Oper. Syst. Rev.*, 40(1):49–61, 2006.

[59] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, January 1965.

[60] T. S. E. Ng and H. Zhang. A network positioning system for the Internet. In *Proceedings of USENIX Annual Technical Conference*, Boston, MA, USA, 2004. USENIX Association.

[61] T. S. E. Ng and H. Zhang. Predicting Internet network distance with coordinates-based approaches. In *Proc. of INFOCOM*, New York, NY, USA, June 2002.

[62] K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the differentiated services field (DS field) in the IPv4 and IPv6 headers. RFC 2474, December 1998. http://tools.ietf.org/html/rfc2474.

[63] D. Oran. OSI IS-IS intra-domain routing protocol. RFC 1142, February 1990. http://tools.ietf.org/html/rfc1142.

[64] *P2PSim: A simulator for peer-to-peer protocols*. http://www.pdos.lcs.mit.edu/p2psim/index.html.

[65] V. Paxson. End-to-end routing behavior in the Internet. In *Proceedings of the ACM SIGCOMM '96*, Palo Alto, California, United States, 1996. ACM.

[66] V. Paxson. End-to-end Internet packet dynamics. In *Proceedings of the ACM SIGCOMM '97*, Cannes, France, 1997. ACM.

[67] *Pepito: A data mining software*. http://www.pepite.be.

[68] M. Pias, J. Crowcroft, S. Wilbur, T. Harris, and S. Bhatti. Lighthouses for scalable distributed location. In *Proc. of the Second International Workshop on Peer-to-Peer Systems*, Berkeley, CA, USA, February 2003.

[69] *PlanetLab: An open platform for developing, deploying, and accessing planetary-scale services*. http://www.planet-lab.org.

[70] S. Qazi and T. Moors. Scalable resilient overlay networks using destination-guided detouring. In *Proceedings of ICC'07*, pages 428–434, 2007.

[71] V. Ramasubramanian, D. Malkhi, F. Kuhn, I. Abraham, M. Balakrishnan, A. Gupta, and A. Akella. A unified network coordinate system for bandwidth and latency. Technical report, Microsoft Research, 2008.

[72] Y. Rekhter, T. Li, and S. Hares. A border gateway protocol 4 (BGP4). RFC 4271, January 2006. http://tools.ietf.org/html/rfc4271.

[73] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan. Detour: Informed Internet routing and transport. *IEEE Micro*, 19(1):50–59, 1999.

[74] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson. The end-to-end effects of Internet path selection. *SIGCOMM Comput. Commun. Rev.*, 29(4):289–299, 1999.

[75] Y. Shavitt and T. Tankel. Big-bang simulation for embedding network distances in Euclidean space. In *Proc. of IEEE Infocom 03*, San Francisco, CA, USA, March 2003.

[76] Y. Shavitt and T. Tankel. On the curvature of the Internet and its usage for overlay construction and distance estimation. In *Proc. of IEEE Infocom 04*, Hong Kong, March 2004.

[77] D. Sontag, Y. Zhang, A. Phanishayee, D. G. Andersen, and K. D. Scaling all-pairs overlay routing. In *Proceedings of CoNEXT*, Rome, Italy, December 2009.

[78] L. Subramanian, I. Stoica, H. Balakrishnan, and R. Katz. OverQoS: an overlay based architecture for enhancing Internet QoS. In *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, San Francisco, California, USA, 2004. USENIX Association.

[79] L. Tang and M. Crovella. Virtual landmarks for the Internet. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement (IMC)*, Miami Beach, FL, USA, October 2003. ACM.

[80] *Vuze (Azureus)*. http://azureus.sourceforge.net/index.php.

[81] G. Wang, B. Zhang, and T. S. E. Ng. Towards network triangle inequality violation aware distributed systems. In *Proc. of the 7th ACM SIGCOMM IMC*, New York, NY, USA, 2007.

[82] X. Wang, Y. Chen, B. Deng, and X. Li. Nonlinear modeling of the Internet delay structure. In *Proc. of ACM CoNext Student Workshop*, Madrid, Spain, December 2008.

[83] B. Wong, A. Slivkins, and E. Sirer. Meridian: A lightweight network location service without virtual coordinates. In *Proc. of the ACM SIGCOMM*, August 2005.

[84] Y. Zhang and N. Duffield. On the constancy of Internet path properties. In *IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pages 197–211, San Francisco, California, USA, 2001. ACM.

[85] H. Zheng, E. K. Lua, M. Pias, and T. G. Griffin. Internet routing policies and round-trip-times. In *Proc. of PAM*, LNCS 3431, Boston, MA, USA, March 2005.

Index