Technische Universität München
Fakultät für Informatik

Diplomarbeit in Informatik

Proximity Neighbor Selection and Proximity Route Selection for the Overlay-Network IGOR

—

Proximity Neighbor Selection und Proximity Route Selection für das Overlay-Netzwerk IGOR

von Cand.-Inf. Yves Philippe Kising

Aufgabensteller:	Prof. Dr. Thomas Fuhrmann (Lehrstuhl für Netzwerkarchitekturen, Technische Universität München)
Betreuer:	Dipl.-Ing. Kendy Kutzner (System Architecture Group, Universität Karlsruhe (TH))
Beginn:	15.12.2006
Abgabe:	15.06.2007

Eidesstattliche Erklärung

Ich versichere, dass ich diese Diplomarbeit selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Yves Philippe Kising

Abstract

Unfortunately, from all known "Distributed Hash Table"-based overlay networks only a few of them relate to proximity in terms of latency. So a query routing can come with high latency when very distant hops[1] are used. One can imagine hops are from one continent to the other in terms of here and back. Thereby it is possible that the target node is located close to the requesting node. Such cases increase query latency to a great extent and are responsible for performance bottlenecks of a query routing.

There exist two main strategies to reduce latency in the query routing process: Proximity Neighbor Selection and Proximity Route Selection. As a new proposal of PNS for the IGOR [1], [2], [3] overlay network, Merivaldi is developed. Merivaldi represents a combination of two basic ideas: The first idea is the Meridian [4] framework[2] and its Closest-Node-Discovery without synthetic coordinates. The second idea is Vivaldi [5], a distributed algorithm for predicting Internet latency between arbitrary Internet hosts. Merivaldi is quite similar to Meridian. It differs in using no direct Round Trip Time measurements like Meridian does to obtain latency characteristics between hosts. Merivaldi obtains latency characteristics of nodes using the latency prediction derived from the Vivaldi-coordinates. A Merivaldi-node forms exponentially growing latency-rings, i.e., the rings correspond to latency distances to the Merivaldi-node itself. In these rings node-references[3] are inserted with regard to their latency characteristics. These node-references are obtained through a special protocol. A Merivaldi-node finds latency-closest nodes through periodic querying its ring-members for closer nodes. If a closer node is found by a ring-member the query is forwarded to this one until no closer one can be found. The closest on this way reports itself to the Merivaldi-node.

Exemplary analysis show that Merivaldi means only a modest burden for the network. Merivaldi uses $O(log\ N)$ CND-hops[4] at maximum to recognize a closest node, where N is the number of nodes. Empirical tests demonstrate this analysis. Analysis shows, the overhead for a Merivaldi-node is modest. It is shown that Merivaldi's Vivaldi works with high quality with the used PING-message[5] type.

[1] A hop means the jump from one node to the other.

[2] Here a framework is a collection of certain procedures to solve problems.

[3] Here, a node-reference means not a direct node but only a "pointer" to a node.

[4] Merivaldi uses a Closest-Node-Job-messages (CND-messages) to obtain closer nodes, where N is the number of participating nodes. Such a message usually needs some hops until its target is reached

[5] The IGOR overlay network uses PING-messages to keep TCP-connections alive.

Kurzfassung

Unglücklicherweise beziehen sich nur ein paar der bekannten DHT-basierten Overlay Networks auch auf die Latenz-Nähe. Somit kann eine Anfrage hohe Latenz aufweisen, wenn über weit voneinander entfernte hops[6] gerouted wird. Es können dabei hops von einem Kontinent zum anderen und wieder zurück vorkommen. Es besteht die Möglichkeit, dass der Zielknoten sich aber letztendlich auf dem selben Kontinent, ganz in der Naehe des anfragenden Knotens befindet. Dadurch wird die Anfrage-Latenz stark erhöht.

Es existieren zwei Hauptstrategien um die Anfrage-Latenz zu verringern: Proximity Neighbor Selection and Proximity Route Selection. Merivaldi, ein neuer PNS[7]-Ansatz für das IGOR [1], [2], [3] Overlay Network wird in dieser Arbeit vorgestellt. Merivaldi verbindet zwei grundsätzlichen Ideen: Die eine Idee ist Meridian [4], ein Framework[8] und sein Closest-Node-Discovery Algorithmus ohne die Verwendung von künstlichen Koordinaten. Die andere Idee ist Vivaldi [5], ein verteilter Algorithmus um Internet Lantenzen zwischen beliebigen Internet Hosts vorhersagen zu können. Merivaldi is Meridian ziemlich ähnlich. Meridian verwendetet direkte Round Trip Time Messungen um Latenzen von Hosts zu erhalten. Merivaldi erhält diese Werte, indem es Latenzvorhersagen verwendet, die von Vivaldi-Koordinaten abgeleitet sind.

Ein Merivaldi-Knoten bildet Ringe, welche exponentiell grösser werdenden Latenzen zu sich selbst entsprechen und fügt in diese Ringe Knoten-Referenzen[9] anhand deren Latenzen ein. Diese Knoten-Referenzen erhält ein Merivaldi-Knoten durch ein bestimmtes Protokol. Ein Merivaldi-Knoten frägt seine Ringmitglieder periodisch nach näheren Knoten ab. Findet ein Ringmitglied einen näheren Knoten, so wird die Anfrage an diesen weitergeleitet. Das passiert so lange bis kein näherer Knoten mehr gefunden wird. Der Knoten, der am nahesten ist, teilt sich dem Merivaldi-Knoten mit.

Eine beispielhafte Analyse zeigt, dass Merivaldi das Netzwerk nur geringfügig belastet. Merivaldi verwendet maximal $O(log\ N)$ CND-hops[10] um einen nahesten Knoten zu finden. Dabei ist N die Knotenanzahl. Empirische Tests untermauern die Ergebnisse der Analyse. Obendrein wird gezeigt, dass Merivaldi auch für den Knoten selbst moderate Komplexität aufweist. Es wird gezeigt, dass Merivaldi's Vivaldi in Verwendung des PING-message[11] Nachrichtentyps, mit hoher Qualität arbeitet.

[6] Ein hop ist ein Sprung von einem Knoten zu einem anderen.

[7] PNS: Proximity Neighbor Selection

[8] Ein Framework bezeichnet hier eine Rahmenstruktur, welche bestimmte Vorgehensweisen beinhaltet um bestimmte Probleme zu lösen.

[9] Eine Knotenreferenz bedeutet hier keinen direkten Knoten, sondern einen "'Zeiger"' auf diesen

[10] Merivaldi verwendet Closest-Node-Job-Nachrichten (CND-messages) um nahe Knoten zu erhalten, wobei N die Anzahl teilnehmender Knoten ist. Solch eine Nachricht benötigt gew"ohnlich ein paar hops bis ihr Ziel erreicht ist

[11] IGOR verwendet den PING-message Nachrichtentyp um TCP-Verbindungen aufrecht zu erhalten.

Contents

Chapter 1

Introduction

During the last years, different proposals of Peer-to-Peer systems have been developed. A promising proposal of Peer-to-Peer systems are the structured overlay networks. In general all structured overlay networks implement a special datastructure, the Distributed Hash Table. This datastructure assigns distinct responsibilities of data items to distinct nodes which participate the overlay network. If one requests for a certain piece of data, the query is routed to the responsible node. With it every participating node stores the pieces of data which the DHT[1] designates to it. If one queries for a certain piece of data, the query is routed to the responsible node. Then this destined node can connect to the requester directly and the desired piece of data can be transmitted.

Usually a requesting node does not know the responsible node before. Therefore the query is routed over some hops[2] until the responsible node is reached. Every routing process ends with a definite result: The desired data item exists or not. Hence, a structured overlay network offers determined query routing. The DHT offers the possibility of determined query routing in $O(log\ N)$ hops, where N is the number of the overlay network participating peers/nodes.

Unfortunately, from all known DHT-based overlay networks only a few relate to the proximity in terms of latency. So a query routing can come with high latency when very distant hops are used. One can imagine hops are from one continent to the other in terms of here and back. Thereby it is possible that the target node is located close to the requesting node.

There exist two main strategies to reduce latency in the query routing process: Proximity Neighbor Selection and Proximity Route Selection. IGOR [1], [2], [3], an overlay network which implements a DHT, is examined positive to implement PNS[3] and PRS[4]. Further-

[1] DHT: Distributed Hash Table
[2] A hop means the jump from one node the other
[3] PNS:Proximity Neighbor Selection
[4] PNS:Proximity Route Selection

more, some different proposals of reducing latency are presented with regard to IGOR. As a new proposal of PNS for the IGOR overlay network, Merivaldi is introduced.

In this diploma thesis the IGOR overlay network is examined to implement a Proximity Neighbor and Route Selection strategy. In this regard a new Proximity Neighbor Selection approach, called Merivaldi, is introduced, analyzed, adapted to IGOR, implemented in IGOR and partially tested with Planetlab [6] on its functionality and effectiveness.

Merivaldi represents a combination of two basic ideas: The one idea is the Meridian [4] framework and its Closest-Node-Discovery without synthetic coordinates and the other idea is Vivaldi [5], [7], a distributed algorithm for predicting Internet latency between arbitrary Internet hosts. A Merivaldi-node forms exponentially growing latency-rings, i.e., the rings correspond to latency distances to the Merivaldi-node itself. In these rings node-references[5] are inserted with regard to their latency characteristics. These node-references are obtained through a special protocol. A Merivaldi-node finds latency-closest nodes through periodic querying its ring-members for closer nodes. If a closer node is found by a ring-member the query is forwarded to this one until no closer one can be found. The closest on this way reports itself to the Merivaldi-node. Exemplary Analysis show that Merivaldi only means a modest burden for the network. Merivaldi uses $O(log\ N)$ CND-hops at maximum to recognize a closest node, where N is the number of participating nodes.

In chapter 2, Peer-to-Peer systems are shortly introduced and compared with Client-Server systems. Additionally the basic idea of the DHT is explained and the importance of proximity consideration is disclosed. Subsequently the IGOR overlay network is outlined.

In chapter 3, I will provide some background of the possibilities in predicting Internet latencies. Especially, it is explained why Vivaldi is preferred in Merivaldi. In consequence, Vivaldi is introduced in detail.

Chapter 4 is dedicated to existing PNS[6] and PRS[7] algorithms. Among others the constraints in implementing PNS and PRS are presented and some different proposals are examined. Especially one which is implemented in Chord [8], a similar overlay network like IGOR. Additionally IGOR is examined positive to implement PNS and PRS.

In chapter 5, the main attention is turned on Merivaldi. One of its basic parts, Meridian [4] is presented. Then Merivaldi's basic idea is introduced and its framework is described.

In chapter 6, IGOR's implementation design with regard to Merivaldi and Merivaldi's implementation itself is shortly presented.

In chapter 7, the test conditions are introduced. The PlanetLab [6] environment is presented shortly and the test results are shown and interpreted.

[5]Here, a node-reference means not a direct node but only a "pointer" to a node.
[6]PNS: Proximity Neighbor Selection
[7]PRS: Proximity Route Selection

Chapter 8 concludes this diploma thesis. Especially Merivaldi, its implementation and the test results are summarized. A short outline of possible future work is given.

Chapter 2

Background

It is getting increasingly obvious that Client-Server based applications can no longer fully meet the evolving requirements of the Internet [9]. More or less this is the reason why distributed systems like Peer-to-Peer networks have emerged during the last years. They provide concepts to overcome the problems based in the Client-Server proposal like resource and performance bottlenecks and single points of failure.

Oram et al. [10] gives a basic definition of what is a Peer-to-Peer system: "A self organizing system of equal, autonomous entities [which] aims for the shared usage of distributed resources in a network environment avoiding central services."

2.1 Peer-to-Peer Systems vs. Client-Server Systems

A Client-Server environment is characterized by the fact that in general one server is a central entity. Only the server is the provider of content or service [9]. Clients can connect themselves to this server to use its services. All in all one can easily assume the Client-Server proposal is a centralized proposal.

In contrast, a distributed system like a Peer-to-Peer system has no central entity. Every node is equal to the other. So this proposal mainly differs from the Client-Server proposal through its decentralized property.

Comparing Peer-to-Peer and Client-Server proposals of distributed systems one can find some advantages and drawbacks for both proposals.

1. **The Client-Server proposal:**

   The drawbacks of the Client-Server proposal.

   - Client-server systems have problems when the number of requesting clients grows. The scaling ability is limited because only a finite number of parallel

requests can be handled at the same time without performance bottlenecks like, e.g., bandwith availability [9].

- Single point of failure: Only one entity is responsible for all services. If the entity breaks down the services are no more reachable [9].
- A service placed on only one server is highly vulnerable for attacks [9].

The advantages of the Client-Server proposal.

- Only one server must be handled and administered.
- The server is the only one to be updated with new information, applications, data, files, and so on.

2. **Peer-to-Peer proposal:**

The drawbacks of the Peer-to-Peer proposal.

- The biggest problem a Peer-to-Peer system faces is where to store and how to find a certain piece of data. Any participating node[1] should be able to retrieve its desired data. The used search procedure for the desired piece of data can increase the query costs to a great extent. For example, flooding is a technique to find out where data items are placed. Sadly, flooding means high bandwith consumption in the network, too. Here, lots of hosts are involved in the search process. But there exist different techniques from proposal to proposal [9], too.
- Versioning: Different versions of files can exist on different nodes. It is difficult to determine which is the newer file and therefore the desired one.
- The desired data can exist and it will not be found anyway. The reason is only a certain search depth is allowed. Especially this is a drawback of unstructured Peer-to-Peer systems [9].
- Range queries are difficult to be performed. More than just one node can hold data within the desired range.

The advantages of the Peer-to-Peer proposal.

- There is no single point of failure because every node is equal to the other. No special server must be asked for service or content [9].
- The bandwith bottleneck of the Client-Server proposal is bypassed. Not only a single node is responsible for all the wanted information. Data, applications, files, etc. are distributed most equally over all participating nodes. Hence, these systems do not suffer from single points of failure [9].

[1]The terms "node" and "peer" are used for the same object, a peer instance

- A Peer-to-Peer system provides an important property when it expands. It scales, also when the system grows fast. The more participating nodes exist, the higher is the probability for positive query results.

2.2 Unstructured Peer-to-Peer Systems

Various Peer-to-Peer proposals exist which can be distinguished in unstructured and structured Peer-to-Peer systems.

Based on how nodes in a Peer-to-Peer system are linked to each other, one can classify the Peer-to-Peer system as unstructured or structured. An unstructured Peer-to-Peer system is characterized by node-links which are established arbitrarily. If a node wants to find a certain piece of data in the network, searching techniques such as flooding are used [9]. On top of everything not each of these proposals can be called a Peer-to-Peer system by the definition made by Oram et al. [10] in 2 on page 11.

Napster-like: Peer-to-Peer with Central Directory

In a Napster-like Peer-to-Peer network a server works like a directory. Its entries contain the addresses of nodes. This directory provides the detection which one should be contacted when querying for a piece of data. The resulting data transmission is done between the nodes without the directory server. This system is called a centralized Peer-to-Peer network [9].

Gnutella-like: Peer-to-Peer without Central Entity

Gnutella [11] is a system without any central server. A node's query is forwarded directly to its neighbors until a certain search-depth is reached. Hence, not every query is successfull even if the desired data exists. Another version of forwarding queries is the random walk. Here, messages are forwarded to one neighbor only. The versions of searching techniques Gnutella uses, can differ [9].

Hybrids

Also there exist combinations of the mentioned proposals. These are called hybrid Peer-to-Peer networks [9].

2.3 Structured Peer-to-Peer Systems

In contrast to the unstructured Peer-to-Peer systems, structured Peer-to-Peer systems use a different strategy. A globally consistent protocol is employed to ensure that every node can route a query to a node that is responsible for the desired data. Another difference is that a node does not randomly select its neighbors.

The most associated type of a structured Peer-to-Peer system are systems which implement the Distributed Hash Table [9]. A variant of Consistent Hashing [12] is used to assign responsibility of certain data items to a particular node. This is similar to the way a traditional hash table assigns keys to particular hash slots [9], [8], [13], [14], [15], [16].

In more detail, a DHT[2] manages data by distributing it across a number of nodes. Also, it implements an efficient routing scheme. Further, DHTs introduce new address-spaces where data items and nodes are mapped. The mapping is done on the basis of their identifiers. In the most cases such an id[3] consists of a large integer value. The id is the execution result of a collision resistant hash function. This hash function is performed with ,e.g., for nodes the IP-address or the filename for data as argument. Peer-to-Peer applications are free to determine this id for the node. But in general there is a collision resistant hash function used. With it a different id of the same id-space[4] is associated to each piece of data and participating node. Such an id illustrates an unique address.

A different contiguous portion of these ids is associated to each participating node. Hence, each of the nodes is responsible for a contiguous portion of the address-space. Mostly a node is responsible for its closest ids as shown in Figure 2.1 on page 14.

In the following it is distinguished between identifiers for nodes and identifiers for data items. “Identifier” is used as usual for both, nodes or data items, “key” is used only for the identifier of a data item.

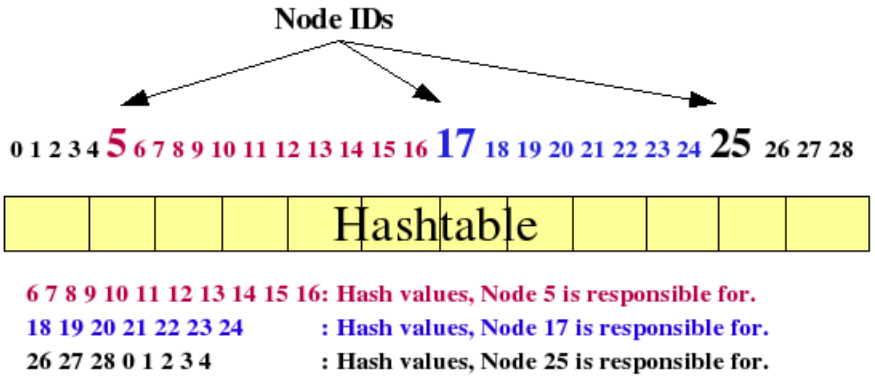

Figure 2.1: A Distributed Hash Table. As one can recognize, the id-space is finite. For example, Node 25 is responsible for the keys 26 to 28 and in addition for keys 0 to 4. This is illustrated through same colored numbers. The mathematic operations are performed modulo 29. Hence, one can imagine a virtual circle.

[2]Distributed Hash Table
[3]id: identifier
[4]id-space: range of identifiers

For instance, as mentioned earlier, the computed key is the result of hashing the filename or the whole file. Then the piece of data is stored at the node which is responsible for this portion of address-space.

The main operation provided by a DHT-based overlay network is the so-called lookup[5] function. This function determines the responsible node for the desired data. More precisely, if a node is requested for a certain data item, it computes the hash function, e.g., of the data's file-name. Then the node routes the query to a neighboring node which is in a way (in the most cases numerically) closer then itself. The operation finishes when the responsible node is found.

The way of routing the query, until the responsible node is reached, is different from DHT to DHT proposal. Moreover, DHT proposals mainly differ in how they manage and partition their address-spaces. With some imagination, these schemes can be interpreted in geometric figures. Topologies like ring, torus, hypercube, tree or butterfly can be found in systems like Chord [8], CAN [13], Pastry [14], Tapestry [15] and Viceroy [16].

Chord as an example: All mathematical operations on the address-space are performed modulo. So a virtual ring-topology is imaginable [8]. The Figure 2.2 on page 18 illustrates this condition.

Minimizing the hop-count of the lookup operation is a great challenge for research. The idea is that every participating node manages a so-called finger table. This finger table is nothing more than a routing table. Such a routing table contains the nodes' ids combined with their transport-addresses. The entries are selected so that a lookup hop-count of $O(log\ N)$ can be realized, where N is the number of nodes which participate in the Peer-to-Peer system.

So when a query is received, the node firstly determines its responsibility for the requested id or key. If it is not responsible, it browses it's finger table to get the next hop. Its selected hop is the id-closest to the requested id or key.

Because the finger table contains node-ids which are at an exponential increasing id-distance[6] away from the node's id itself, each hop reduces the id-distance to the target node by half. Therefore an average lookup hop-count of $O(log\ N)$ can be realized from most DHT-based overlay networks [9].

The field of applications for such DHT-based overlay networks are distributed applications which want to profit from distribution and scalability. Some examples are:

- Distributed file systems like IgorFS [1],
- Distributed video recorder like VIDEGOR [17],
- Distributed back-up systems like OceanStore [18],

[5]Query or lookup have the same meaning.
[6]id-distance: identifier distance

- Distributed systems to provide load balancing
- and so on.

Although the performance of all of these applications strongly depends on the speed of the lookup function, latency is almost out of consideration in the development of DHT-based overlay networks. The lookup hop-count is optimized to $O(log\ N)$. But mostly this will not result in a low latency lookup, too.

So if hops of great distance are used the latency of a query routing can increase dramatically. One can imagine hops are from one continent to the other by the meaning of here and back. Thereby it is possible that the target hop is located close to the requesting node. Such cases increase query latency to a great extent. Hence, this can decrease the performance of a query. This can be avoided with intelligent neighbor selection or route selection. Neighbor selection means filling the finger table with latency close nodes. This avoids high latency fingers in the routing table. Route selection means query forwarding to a node which is the closest in latency. This tries to avoid high latency hops altogether. Additionally a combination of both is possible. Using these proposals, the routing process uses nearest neighbors which can satisfy a routing progress. Therefore the lookup delay can be decreased and Proximity Neighbor Selection and Proximity Route Selection should be implemented.

In this diploma thesis the IGOR overlay network is examined to implement a Proximity Neighbor and Route Selection strategy. In this regard a new Proximity Neighbor Selection approach, called Merivaldi, is introduced, analyzed, adapted to IGOR, implemented in IGOR and partially tested with Planetlab[6] on its functionality and effectiveness. IGOR is presented in the following section 2.4 on page 17.

2.4 IGOR

The IGOR [1],[2],[3] protocol specifies a structured overlay network on the top of TCP/IP[7] in the ISO/OSI[8] stack. Hence, IGOR uses TCP's reliability, flow control and congestion control. Within the TCP connections, control messages for IGOR and data messages for applications built on IGOR, are sent.

IGOR offers service oriented message delivery, i.e., different applications can use the same IGOR overlay for their different purposes. IGOR distinguishes applications through application-ids[9].

Like other DHT-based overlay networks, IGOR provides a lookup function to find pieces of data. This function aims to identify the node which holds the desired information, ,i.e., the key-value pair. Like usual in a DHT-based overlay network, a certain amount of participating nodes is queried for this search process. IGOR tries to speed up this operation by minimizing the hop-count like the most DHT-based overlay networks do. Like the other DHT-based overlay networks this is achieved with the earlier mentioned routing tables.

Other functions provide the perpetuation and consistency of the overlay network.

Further, IGOR extends the idea of consistent hashing [12],

- Addition or removal of a node does not require significant changes in the key distribution.
- Addition or removal of a node does not require significant remapping.
- All nodes are responsible for roughly the same number of keys. Hence, load balance is achieved.

Unlike Chord [8], IGOR implements a symmetric routing scheme. An IGOR node is not only responsible for the direct succeeding ids but also for the direct preceding ids. Additionally the distance between two ids[10] is determined not only by one direction as shown in Figure 2.3 on page 20 and the following sections.

2.4.1 IGOR-Operation Mode

The IGOR protocol specifies where node-ids and data-ids[11] can be found, how new nodes can be added and how the nodes have to behave if they leave the overlay network. In

[7] TCP/IP: Transmission Control Protocol/Internet Protocol

[8] The Open Systems Interconnection Basic Reference Model. For more detail: http://en.wikipedia.org/wiki/ISO/OSI

[9] application-ids: application ids

[10] id-distance: The numerical distance between two ids

[11] Data-id: The identifier a piece of data is associated with

addition, IGOR satisfies the necessity of consistency and perpetuation of the network with periodic performed algorithms.

Nodes and objects (data items) obtain an unique id. This is achieved with a hash function which is known to all participating nodes. This function produces the ids of the nodes and objects. Its argument can be in case of a node, for example, the IP-address. In the case of an object the name or something else. An important condition is that always the same length for an id is used.

IGOR uses ids with a size of 160 bits. Hence, 2^{160} different ids exist. Now by the use of these ids every node or object can be placed at an unique position in the system. When the ids are ordered clockwise, it is easy to imagine that a virtual ring structure *modulo* 2^m is formed (m is the id-length). See Figure 2.2 on page 18 for more clearness. The first and the last node-id are connected because of the modulo-operation.

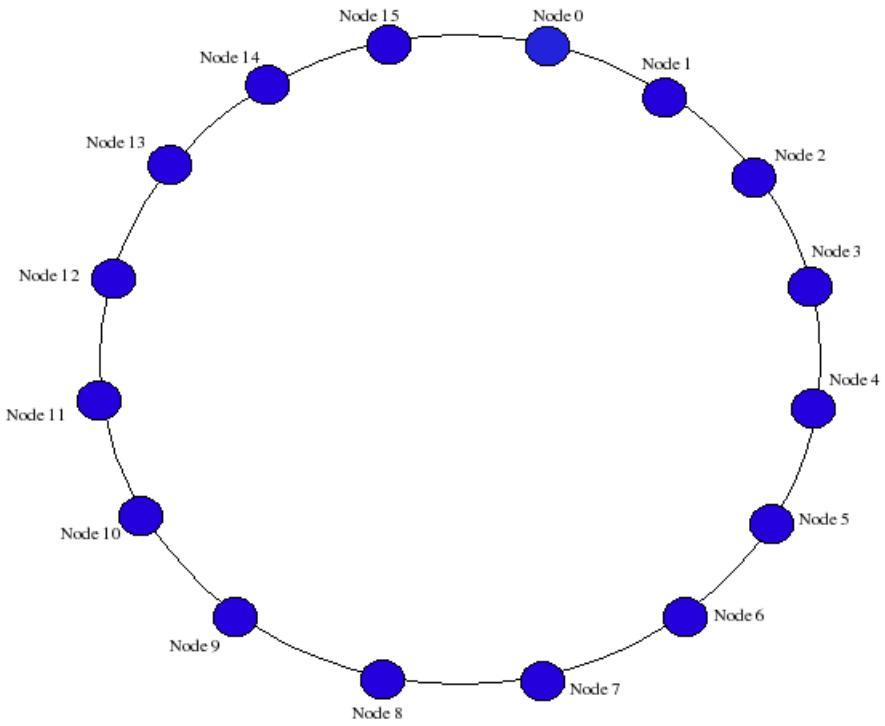

Figure 2.2: The virtual ring structure with 16 participating nodes.

IGOR declares nodes to be responsible for certain ids unlike other well known DHT-based

overlays like, e.g., Chord. IGOR implements a symmetric id-responsibility distribution whereas Chord implements an asymmetric one. In Chord, the ids between two direct neighboring nodes are fully assigned only to one of the both [8]. This condition is illustrated also in Figure 2.1 on page 14. In contrast, the symmetric distribution divides the key space between two direct neighboring nodes. See Figure 2.3 on page 20 for more detail.

Finding the responsible node needs a certain processing. In a simple version, such a navigation within the ring geometry progresses because every node knows its direct neighbors. If a node is requested for a certain id, the node decides which one of its neighbors is closest. In this way, the query is routed until its target is reached.

This way of processing a query is therefore very slow and does not scale well. As one can imagine, in the worst case one have $O(N)$ nodes on the path to the target, where N is the number of participating nodes. So the mentionend idea of finger tables is included in IGOR. With those an average lookup hop-count of $O(log\ N)$ can be achieved, where N is the number of participating nodes.

The size of the finger table assumes the number of nodes an IGOR node needs to know. This is $O(log\ N)$, where N is the number of participating nodes. So IGOR scales quite well when the system grows.

2.4.2 Finger Table

The idea of navigation in a ring structure only with the knowledge of the predecessor and successor is not very efficient as denoted above. Imagine that half of the ring must be walked through.

To improve this, the finger tables are integrated. A finger table is nothing more than a routing table. Mostly, it contains pairs of node-ids and transport-addresses.

Such a finger table consists of m entries on average, called fingers. The size is directly dependent on the size of the system. IGOR has at most 160 entries, usually $log_2\ N$, where N is the number of participating nodes.

This finger table is used to reduce the average hop-count a lookup operation usually needs. The node where the query is routed, is the finger whose id is the closest to the requested id. On its way a query routing normally reduces the id-distance between the desired key and the target each hop by half.

The finger table must be filled intelligently to reach an average lookup hop-count of $O(log\ N)$ for any query, where N is the number of participating nodes. In the following it is explained how IGOR fills its finger table: The nodes which are interesting for a certain entry can be found in a certain id-distance interval to the node's id itself. First of all the id-distance between two nodes n and i is defined as follows:

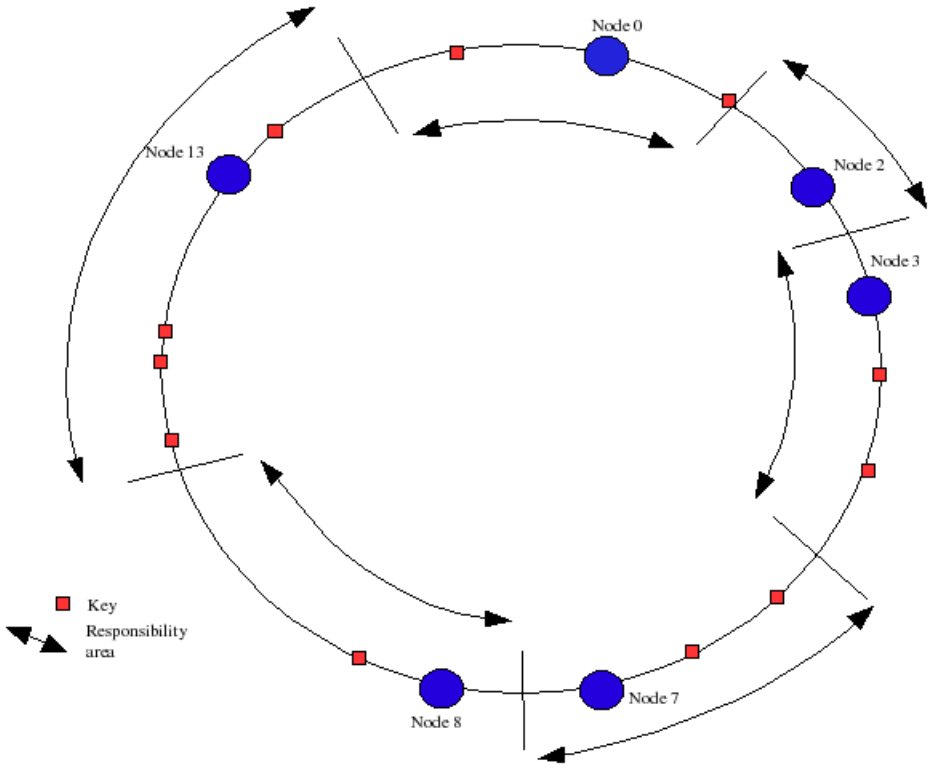

Figure 2.3: The responsibility areas in an IGOR ring structure are divided between direct neighboring nodes. Nodes are illustrated as circles. Data items as squares.

$$d(n,i) = |n - i| \ mod \ 2^m \tag{2.1}$$

The i-th finger of node n is now defined as the id-distance between node n and its i-th finger. As one can see the i-th finger can be a node whose distance to n is within a certain range:

$$d(n, finger(i)) \in [2^i, 2^{i+1}] \ with \ m \ < \ i \ \leq \ 0 \tag{2.2}$$

The id-distance between the node n and the finger i grows exponentially. The id-distance between directly succeeding fingers should be constant. Every i-th finger is a logarithm of base 2 away from its direct successor, the $i + 1$-th finger or predecessor $i - 1$-th finger.

2.4.3 Key Lookup

The basic property of structured Peer-to-Peer systems like IGOR is the lookup function for an id-value pair. That means to locate the desired piece of data or information. This operation is performed in IGOR like it is explained in the following and shown as an example in Figure 2.4 on page 21. First an arbitrary node is asked for an id. This processing is called a query for an id . The query is masked by a message with a target. The target is the identifier of the desired piece of data or node.

Firstly the requested node checks its responsibility for the target id. If this is not the case it browses its finger table and extracts the finger which is the closest one in terms of id-distance to the target of the message.

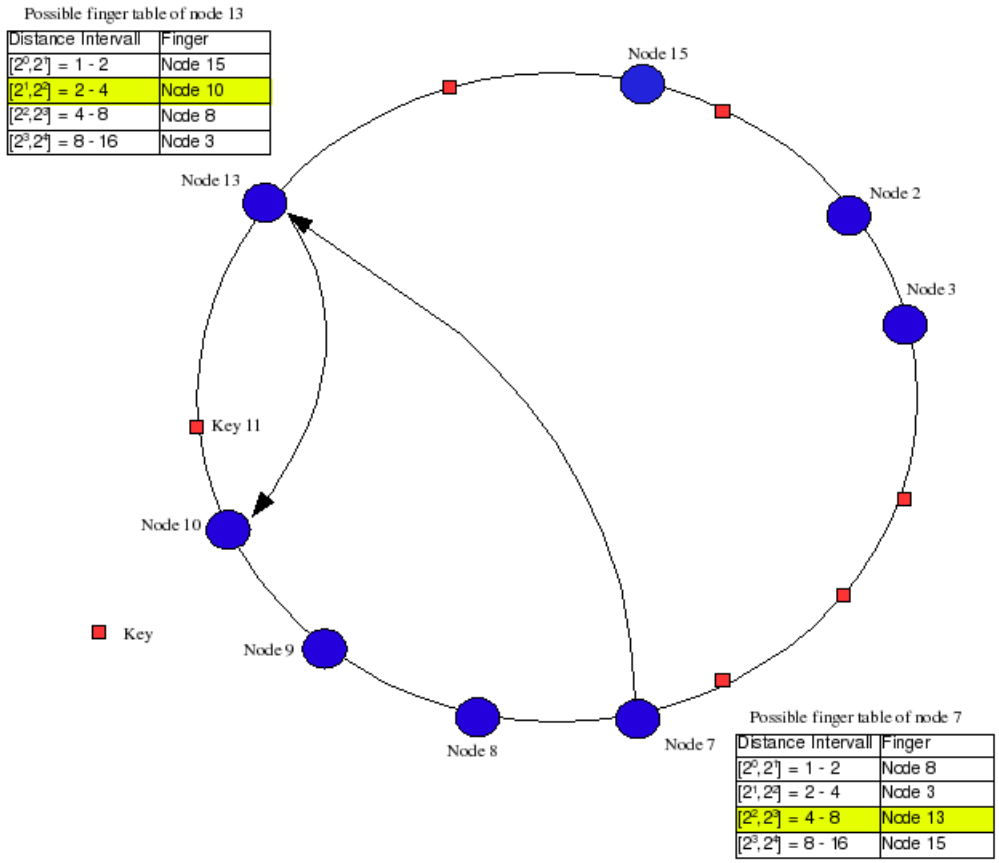

Figure 2.4: Identifier lookup. Node 7 is asked for id 11. The displayed finger tables contain possible fingers. After node 7 has searched for the closest finger it forwards the query towards node 13. Node 13 searches its finger table and forwards the query to node 10. Node 10 is responsible for the id and answers directly to the requester.

Then it forwards the message to this finger. If the requested node is responsible, it reports itself directly. In this way a direct TCP-connection can be established to transmit the desired piece of data.

2.4.4 Node Join

A node which wants to participate in IGOR must be aware of a bootstrap node. This bootstrap node has to participate in IGOR already.

There are some things a new node must obtain to become a full member of the network. The new node creates an id, which is now its id in the IGOR network. After that, the new node, let me call it "newbie", will insert the bootstrap node into its finger table.

Now newbie queries the bootstrap node for its own id. The reason for that procedure is that newbie wants to know about nodes which are nearby itself in the id-space. In that manner newbie gets knowledge about its direct neighbors in both directions.

Newbie obtains some not already known nodes from the bootstrap node. Newbie can order the bootstrap node to let it know about nodes which are near its id with growing id-distance. Newbie now begins to handle the new nodes. It decides if the new nodes are suitable for its finger table. Additonally these new nodes can be asked for new nodes again. As mentioned in the first part of this section, newbie starts some periodic activity similar to this. This is done to maintain the consistency of its finger table.

2.4.5 Stabilization

A big overlay network with many participants is characterized by a high level of addition and subtraction of nodes. To overcome this nature of Peer-to-Peer systems the finger tables must be updated to guarantee a stable lookup performance in terms of hop-count.

Hence, an IGOR node performs two periodic updates of its finger table: One to update its direct neighbors in id-space and one to obtain neighbors with exponentially growing id-distance to itself.

Therefore two functions exist. One is for getting the direct neighbors in id-space. This one is called $stabilize()$. The other function $fixfinger()$ is to obtain neighbors with growing id-distance as mentioned above. This function guarantees the equal distribution of fingers as mentioned under 2.4.2 on page 19 and illustrated in Figure 4.3 on page 42.

These two functions perform and use special maintenance messages. Certain nodes are requested within these messages through the message's target. Every node which receives such a message searches its finger table for a certain number of nodes which are the closest to the messages' target. The id-transport-address pairs of these closest nodes are placed

in the answer message. When the requesting node receives the answer it can examine if the new ids are useful for its finger distribution.

Stabilize() requests just for node-ids which are the nearest to its own id. Therefore the message's target is its own id. With this function the node tries to obtain its id-closest neighbors.

$Fixfinger()$ requests for node-ids which are useful in the distribution of fingers. So it looks for nodes which are a certain id-distance away from its own. Therefore the message's target is an id which is an exponentially growing id-distance away from its own. The reason for the exponential increasing id-distance is explained in 2.4.2 on page 19.

2.4.6 Node drop out

When a node leaves the system there is usually no announcement. For this account the mentioned functions are responsible to stabilize the network. When an IGOR node shuts down it just closes all connections to itself. It stops its regular tasks. The network stabilizes itself with the earlier mentioned functions.

2.4.7 Complexity and Load balancing

IGOR uses the idea of Consistent Hashing [12] combined with routing tables. Consistent Hashing has certain advantages:

- The addition and the removal of one node does not change the distribution of ids in a hard way.
- No need for significant remapping of ids to nodes.
- Every node is responsible for roughly the same number of keys.
- Keys are only remapped to new nodes to preserve the consistency of the network.

Because of the equal distribution of node-ids load balance is achieved. As mentioned earlier in 2.4.2 on page 19 , IGOR offers a query for an id (lookup) within $O(log\ N)$, where N is the number of participating nodes.

Chapter 3

Prediction of Internet Latency

Nearly all PNS and PRS proposals use latency prediction to improve lookup performance. Predicting Internet latency between arbitrary Internet hosts provides a big advantage. One can estimate the latency without costly prior communication. In this diploma thesis a new PNS proposal, called Merivaldi, is developed. Merivaldi which will be introduced in 5 on page 43, uses latency prediction, too. To estimate or predict Internet latency between arbitrary Internet hosts different proposals have been developed during the last years. Nearly all of them pursue a similar approach: Computing and assigning synthetic coordinates to Internet hosts with regard to their latency characteristics.

These latency prediction-proposals' great advantage can be found within their prediction property: No previous communication is necessary to obtain latency characteristics and with it, no additional time loss is realized.

In the following some proposals of predicting Internet latency are shortly introduced:

IDMaps [19] uses thousands of so-called "Tracer-Nodes". Each "Tracer-Node" is aware of the latency to each other "Tracer-Node". Additonally a unique assignment of "Tracer-Nodes" to CIDR[1]-address-prefixes is done. The latency between those CIDR-prefixes and the "Prefix-Tracer-Nodes" is known. Hence, the latency between two nodes is the sum of the latencies of the two "Prefix-Tracer-Nodes", one for each node and the latency between the "Tracer-Nodes" themselves.

GNP [20] One possibility to compute coordinates which assign latency is to use several nodes, so-called landmarks. In GNP a small number of hosts are chosen as landmarks. As the landmark coordinates have been committed, the non-landmark nodes can determine their coordinates by measuring their distance to each of the landmarks. All in all the necessity of a fixed infrastructure is the main drawback of this proposal.

King [21] method measures the latency between two DNS[2] servers. It is assumed that

[1] CIDR: Classless Inter Domain Routing
[2] DNS: Domain Name System

most of the end hosts are located close to their DNS name servers and recursive DNS queries can be used. The resulting RTTs[3] between these pairs of DNS servers can be used as estimates for the latency between these end hosts. In general, the King method is used for collecting data for network simulators to map a realistic latency distribution. Because a lot of peers can be found behind one DNS server it is not suitable for a Peer-to-Peer network to determine latency closest nodes.

Vivaldi [5], [22] is a fully decentralized algorithm for predicting Internet latency. Therefore it is the most suitable approach for lots of Peer-to-Peer applications. Vivaldi is selected and implemented in IGOR as a part of Merivaldi. Vivaldi is explained in more detail in the following section in 3.1 on page 25.

3.1 Vivaldi: Distributed Prediction of Internet Latency

One established procedure of predicting Internet latency between arbitrary Internet hosts is an algorithm called Vivaldi[5], [22] . Vivaldi is a fully distributed algorithm which assigns synthetic coordinates to Internet hosts. The coordinates are assigned in a way that the euclidean distance between the coordinates predicts the latency between these hosts. The algorithm was developed by Robert Cox et. al [22], [8] for Chord to avoid contacting distant hosts when performing data queries.

The basic idea is to map the learning problem, assigning accurate coordinates to hosts, to a physical spring network. The forces of the springs care for the correct adjustment of the network, i.e., the forces care about the accurate adjustment of synthetic coordinates. How the forces try to move a nodes' coordinate is illustrated in Figure 3.1 on page 28.

The concept behind Vivaldi is that springs are placed between each pair of nodes. The rest length of such a spring is set to the latency. The length of each spring of a node pair determines the distance between the current coordinates of these two nodes. The potential energy of a spring is proportional to the displacement from its rest length. This displacement now is identical to the prediction error of the coordinates. All in all minimizing the prediction error is equal to minimize the potential energy [5], [7].

The challenges one meets by implementing the Vivaldi algorithm are:

1. **Model selection**. Getting the best space-model with lowest error , i.e., which space-model maps the latency distribution of the Internet without high failure is a non-trivial question. The main problem is almost every metric space satisfies the triangle inequality. The direct path from A to B is shorter or equal to the indirect path over intermediate nodes. In some cases the Internet violates this triangle inequality.

[3]RTT: round trip time

Fortunately these violations are rare [23], [20]. Vivaldi's prediction error, using an 3-dimensional euclidean coordinate space, is competitive to that one of GNP [5].

Obviously hosts are distributed over the globe. One can assume their synthetic coordinates can be assigned with regard to their geographic distance. For example, for a node placed in Europe and its neighbor placed in the United States of America latency is greater than if they both were positioned in Europe. It is imaginable that the latency distribution fits the geographic distribution quite well. All in all this consideration leads to the assumption that a coordinate space of two dimensions should be sufficient.

But, as announced, unfortunately, this does not maps the reality at all. The reasons are access-link delays, packets often take inefficient routes as they move from one to the next backbone and different circuits offer different characteristics. So this easy picture cannot constitute the real world latency distribution.

One of the proposed space-models combines height vectors with two dimensional euclidean coordinates. The size of the height vector is captured through the transmission delay of the access links. Latency determination with height vectors just redefines the usual vector operations. But despite this favored space-model, some empirical analysis show when using more than 4-dimensions this height-vector-model can be outperformed a bit[5], [24]. Additonally the height-vector model has the drawback that hosts which are placed in the same Intranet, have to do the access link delay twice to reach each other [5]. Further, to my best knowledge, no analysis with a height-vector model using more than only 2 dimensions exist until now. All in all and especially for implementation reasons the N-dimensional model is preferred by myself. So, it is possible to assign different numbers of dimensions dynamically. But for further work it should be easy to adapt IGOR's Vivaldi regarding new cognitions.

The normal vector operations:

Addition (Subtraction):

$$[x_1, ..., x_n] + [y_1, ..., y_n] = [(x_1 + y_1), ..., (x_n + y_n)] \tag{3.1}$$

Norm:

$$|||[x_1, ..., x_n]||| = \sqrt{x_1^2 + ... + x_2^n} \tag{3.2}$$

Multiplication:

$$\alpha \times [x_1, ..., x_n] = [\alpha x_1, ..., \alpha x_n] \tag{3.3}$$

where n is the n-th dimension.

2. **Scaling** is important. Synthetic coordinates are very useful in large-scaled applications. In large-scaled applications it is no longer feasible to perform direct measurements without intensely increasing costs. Vivaldi is fully distributed and analysis show that only a small finite number of neighbors is sufficient to perform the needed measurements to obtain good results [5]. Therefore one can conclude Vivaldi scales well.

3. **Distribution**. Vivaldi is implemented in a decentralized way, i.e., Vivaldi is a fully distributed procedure which is performed on every node.

4. **Overhead**. Minimize the probe traffic. Vivaldi gathers the needed information directly from the application's existing communication: Vivaldi piggy-backs the needed update information on existing communication units (messages).

5. **Accuracy**. The relative error as the main measure for accuracy illustrates the ratio between the absolute prediction error and the actual network latency. The prediction of latency derived from Vivaldi exhibits median relative errors between 5 to 10 percent using 3 to 5 dimensions on wide area networks [25], [5]. Additionally Vivaldi is robust against high error nodes. Therefore it offers good accuracy under Churn[4], too. Additionally Vivaldi adjusts a node's coordinates reactively whenever a change is detected. Furthermore, Vivaldi's prediction error, when using a 3-dimensional euclidean space, can be classified in the same range as the one of GNP [22] as mentioned earlier.

6. **Changing Network Conditions**. Vivaldi is able to adapt itself to changing network conditions. The algorithm reacts directly on changes: The nodes' coordinates are constantly updated with new measurements. So if the topology changes, Vivaldi nodes will adapt themselves to the new conditions.

One can conclude, Vivaldi addresses all of these design criteria with high quality solutions.

3.1.1 Mode of Operation

As explained earlier through a series of small timesteps the force on each node moves the node's coordinates towards their right position. Hence, on each timestep the force on each node is calculated. Then the node is moved in the direction of the resulting force.

The length of movement is proportional to the applied force and the size of the timestep. In that manner each movement helps to decrease the energy of the system and therefore to reduce the prediction error.

Different versions of Vivaldi are imaginable: A centralized version and a distributed version. The centralized version considers all force-vectors at once. In contrast the decentralized considers only one force-vector at once. The distributed version is to be preferred because

[4]Churn: The steady arrival and the leaving of nodes in a Peer-to-Peer system

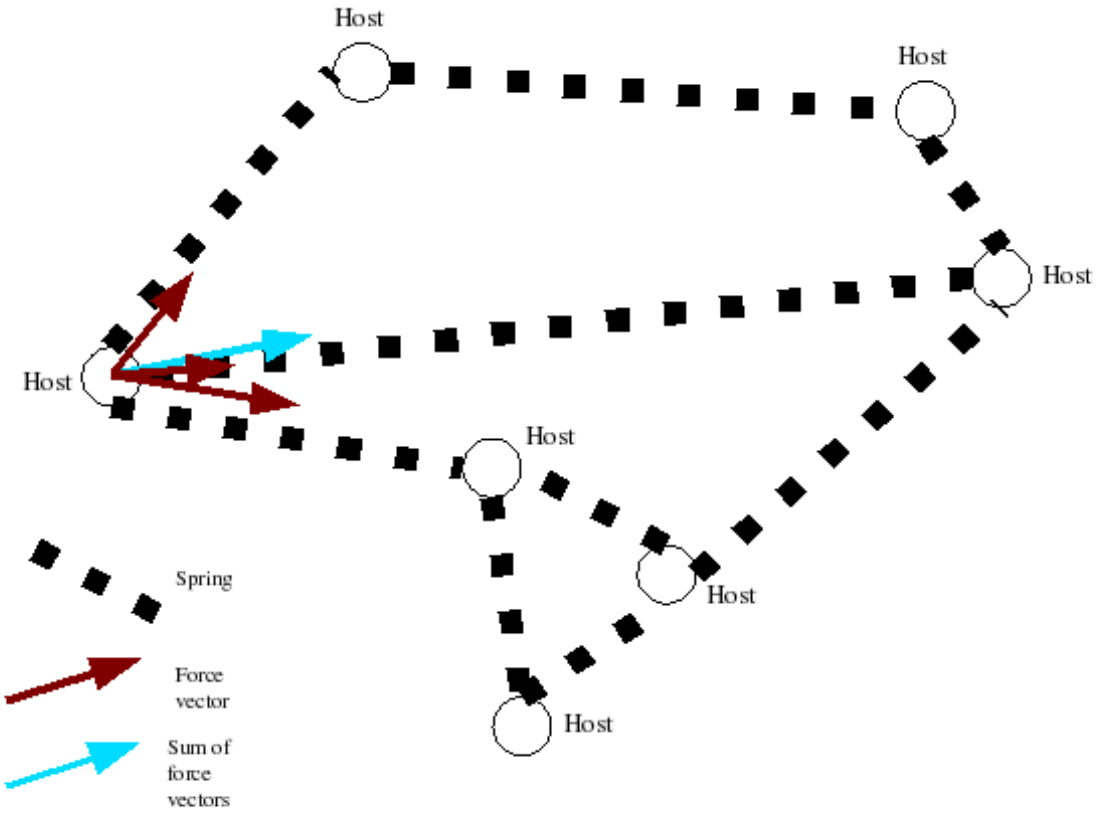

Figure 3.1: Vivaldi: Spring network. Here one host is shown as an example. The forces are illustrated with arrows. The sum of forces shows where to move.

a node can react directly on one measurement. It is not constrained to wait for all other measurements. In the distributed version, whenever a measurement has been made between two nodes, a node adjusts its coordinates to reduce the error between the measurement and the prediction distance. To avoid oscillation only a fraction, δ, of the desired movement is done. This helps to realize the addition of forcevectors in terms of the centralized version. For example, the next measurement affects another node and with it, another direction. δ is the only user-tunable parameter.

A node should initialize δ to a greater value when it starts. A big δ is chosen to move quickly to an accurate position. The lowering of δ is done to avoid the mentioned oscillations.

The main difficulty is to find the best value for δ. Additionally a solution is needed for nodes with high errors in their coordinates. Believing the new inaccurate coordinates and with a big δ a node can adapt its own coordinates to a great wrong value. To avoid this, every node has to maintain the accuracy of its own coordinates. This is realized as a moving average of recent relative errors. The determination of δ depends on error determination. See the following section "The Vivaldi Algorithm" for more detail.

3.1.2 The Vivaldi Algorithm

Every Node additionally maintains an error estimate, a moving average of recent relative errors (absolute error divided by actual latency). This variable for this average is e_j. A node J with coordinates x_j and error estimate of e_j has been measured to be RTT ms away. The parameter c_c should be fixed to 0.25. Here the best empirical results have been seen [5].

vivaldi(RTT, x_j, e_j)

- Computing the weight of a sample based on local and remote error is important to determine the timestep δ. Assuming the executing node is on a correct position, the error estimation of the remote node is responsible for the weight. The smaller it is, the higher is the weight of this sample. And therefore the bigger the movement is.
  $w = \frac{e_i}{(e_i+e_j)}$;
- Compute the relative error of this sample. It is necessary to state the relative error of this sample to adjust the weighted moving average of the local error.
  $e_s = \frac{\left|\|x_i-x_j\|-RTT\right|}{RTT}$;
- Update the weighted moving average of the local error. Here the local error is adjusted with regard to the samples' weight. The higher the sample weight is, the more the local error is adjusted.
  $e_i = e_s \times c_e \times w + e_i \times (1 - c_e \times w)$;
- Update the local coordinates. Determine the direction of movement through the unit-vector $u(x_i - x_j)$ and the size of movement through δ, which is dependent on own and remote error.
  $\delta = c_c \times w$;
  $x_i = x_i + \delta \times \left(RTT - \parallel x_i - x_j \parallel\right) \times u(x_i - x_j)$;

[23] ,[5].

Chapter 4

Proximity Route and Proximity Neighbor Selection

It can be a great drawback if DHT-based overlay networks do not exploit network proximity. Nodes which participate in a large scaled Peer-to-Peer network are often placed all over the globe. One can imagine a query for a key is routed from one continent to the other, just to detect that the key is placed on the same continent on a latency-close node. Cases like this increase the latency of a query to a high level. Finally the total latency of a query's routing path should be not more than a small multiple of the underlying Internet latency.

Various proposals have been developed to overcome this duty: Geographic Layout, Proximity Neighbor Selection and Proximity Route Selection are the most known. Obviously they all adopt the same proposal to get the best ratio between progress in id-space and closeness in latency[26]. In this thesis PNS[1] and PRS[2] are explored in detail with regard to the overlay network IGOR.

Geographic Layout. It is tried to place nodes which are nearby in latency, id-close in id-space.

Proximity Neighbor Selection. The entries of the finger tables are done considering their latency. If the possibility is given that there is a choice for an entry in the finger table, the node which features the best latency property is selected [27].

The one constraint is the range of id-space from which the the entries can be selected [28]. Nevertheless when using PNS, the average query hop-count of $O(log\ N)$ stays equal, where N is the number of participating nodes.

Proximity Route Selection. Proximity Route Selection differs from PRS: The entries for the finger table are selected without any consideration of latency. On a query's routing

[1]PNS: Proximity Neighbor Selection
[2]PRS: Proximity Route Selection

path the next hop is chosen not only for doing the best progress towards the key in id-space. The challenge is to find the best ratio between progress in terms of id space and latency minimization. All of the fingers which are id-closer to the queried id are candidates for the next hop. The one with best latency property is selected here. One drawback is that the real hop-count can be increased.

4.1 Preconditions for PNS and PRS

Within this section the conditions needed to implement PNS or PRS are discussed.

4.1.1 Constraints

The crucial question is how well the DHT-based overlay network can adapt to the underlying topology. The better this is possible the better proximity can be exploited. After Gummadi et. al [28], the main constraint which is responsible for the ability to exploit proximity constitutes itself in the flexibility of selecting fingers or and the next hop.

Different DHT-based overlay networks offer different ways of routing and id management. As mentioned in the earlier sections a DHT can be imagined as a geometric figure. Chord routes along a ring structure, IGOR too [8], [3]. The geometry a DHT features depends on how routing is processed and how the key-value pairs are distributed.

One can assume different geometries offer different degrees of flexibility in choosing neighbors and the next hop. This becomes clear if we not only account the typical metrics of concern: The number of neighbors, the average hop-count [28].

But what really decides is the ability to implement PNS and PRS. Therefore the property to select fingers and next hops from certain range is needed. The greater the range of possible neighbors is the better proximity can be exploited [28].

But the potential improvement is limited. A finger is selected of a certain id-distance interval to the IGOR-node. The possible fingers of this id-range are not likely to be among those which are closest in latency [27], [26], too.

Considering a query gets close to the target-id in id-space, the number of remaining nodes which are id-closer to the key decreases with each hop. Thus, the latency increases on average when getting closer to the id-target [29].

4.1.2 Perfect PNS: Identifying the closest Neighbor

Identifying the closest node within a subset of possible nodes can become expensive: The id-range within the finger can be found is growing exponentially. Therefore various heuristics

have been designed and examined. *PNS(K)* is one of the proposed and is defined as follows: The *PNS(K)* algorithm takes the first K nodes within the possible id-range into account. Hence, it samples these ones to determine which of them really exist. The one with the lowest latency is selected as the finger of this id-range.

Ideal PNS refers to *PNS(K)* when K is equal to the total number of nodes of allowed nodes in the id-range. In that case every finger table entry points to the node with the lowest latency [28],[29].

An empirical statement concludes that *PNS(16)* improves latency characteristics [28]. In this case only the latency properties of the first sixteen consecutive nodes in the corresponding ranges are compared. Obviously, if the id-range of possible nodes becomes greater than 16, the closest nodes within this range will be found to a decreasing probability. The advantage this version offers, is a convenient implementation in existing systems. In addition the relative costs of the algorithm are low [29].

4.1.3 Challenges for PNS and PRS

Proximity Route Selection requires no changes in building the routing table. Also no changes in maintenance mechanisms are needed [27]. The next hop is chosen just in time and therefore no previous consideration of proximity is performed.

Proximity Neighbor Selection requires changes in building the routing table. The selection of fingers and the maintenance mechanisms must be adapted [27]. Latency-close nodes should be inserted in the routing table only. Therefore it is implicitly essential to get knowledge about them. Additionally it is necessary to integrate this selection criterion in the maintenance mechanisms to guarantee resilience considering latency-proximity.

4.2 Proposals for PNS and PRS

Various proposals exist for Proximity Routing and Proximity Neighbor Selection. This section primarily presents the most common proposals suitable for ring structures.

4.2.1 Proximity Routing

Some heuristics exist to implement Proximity Routing. These algorithms must handle a tradeoff between hop-count and latency. Without consideration of proximity the next hop is selected in a "greedy way" just to decrease the number of hops.

There exist some heuristics that trade hops for latency [28]:

1. **Usually equal Hop-Count**. The number of routing paths from two ids which are $O(N)$ in id-space away from each other is within $O(log\ N)$, where N is the number of participating nodes. In more detail the first node maintains $log\ N$ fingers which are in the position to decrease the id-distance to the target, where N is the number of participating nodes.

   Now switching to the second node. One have $(log\ N) - 1$ fingers which can decrease the id-distance again. One can conclude $(log\ N)!$ possible routes exist [28], where N is the number of participating nodes. Obviously every neighbor which is closer to the destination is a possible next hop. Imagine the worst case can increase the hop-count to $O(N)$ hops, where N is the number of participating nodes. This happens if the first finger is selected by each hop to forward the query.

   Heuristic results from that fact: The next hop can be chosen from a certain portion of fingers. This portion is called the candidate set. Selecting one of them usually does not increase the hop-count. To select the candidate set, the id-distance to the target is expressed in binary notation. If there is a 1 in the i-th position, finger i is selected. The achieved candidates now can be examined for latency. The closest one is picked as next hop [28].

2. **Without Adherence of Hop-Count**. Another possibility without adherence of hop-count is the following: Taking multiple hops of smaller spans instead of one with large span. For example, take two successive hops with the $(i-1)$-th fingers of span 2^{i-1} each instead of a single hop with the i-th fingers of span 2^i. This is a possibility to enrich the candidate range [28].

4.2.2 Proximity Neighbor Selection

As earlier mentioned in 2.4 on page 17 a node which participates in a DHT-based overlay network like IGOR fills its finger table with fingers from different id-range intervals. The crucial question is how to get knowledge about nodes of which id-distances are within the desired id-ranges. These nodes satisfy the id-distance condition. Here some proposals have been made. Not all of them are suitable for every DHT-based overlay network. The most common are described in the following.

1. **Global Sampling(Tuning)** . When looking for a finger within a certain id-distance, queries for such ids are performed. The answering nodes' ids are almost suitable. The one with the best latency property is the new entry in the finger table within the desired id-distance.

   Gummadi et al. [28] showed that sampling 16 nodes for one entry allows to come close to $Ideal\ PNS$ for small system sizes.

   This proposal provides one big drawback: For instance, two nodes separated from the core Internet by the same high latency access link: The relative high latency seen

by this two nodes makes them attractive fingers for each other. This is illustrated in Figure 4.1 on page 34.

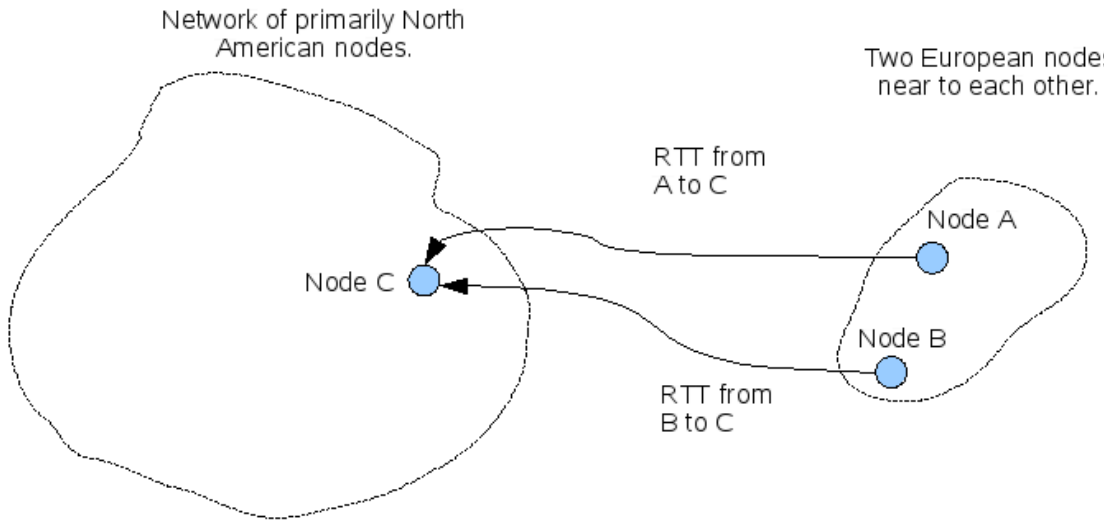

Figure 4.1: Initially, the nodes A and B are unaware of each other. Because of the high latency seen by the core network in North America, they are unattractive fingers for other nodes. In contrast, they are attractive fingers for each other. If both maintain node C as a finger they are able to find each other by asking C for its inverse neighbors [30].

The time global sampling needs to find each other is proportional to the size of the network. Nobody may want them to be an entry in their finger tables caused by their high access link latency. Finally, they may not find each other before their sessions end [30].

2. **Finger's Fingers**. This technique relies on the expectation that proximity in the network is roughly transitive. If a node discovers one nearby finger, this finger's fingers are probably nearby, too. In this way a node can walk through the graph of neighbor links to the set of nearby nodes. Unfortunately, the same drawback is given as mentioned above [30].

3. **Finger's inverse Fingers**. Sampling nodes which have the same fingers. When it is presumed that their fingers are mostly nearby, the above mentioned problem can be solved. Although those two nodes are unlikely to be fingers of many other nodes they are likely to select the same fingers from outside their isolated domain. For this reason they propably find each other in the set of their fingers' inverse fingers [30]. For more clearness see Figure 4.1 on page 34.

4. **Recursive Sampling**. The number of fingers in which inverse finger sets the two isolated fingers can find themselves is very small. Until the two isolated nodes have

found very nearby fingers, they will be unlikely to find each other among their fingers' inverse fingers. To remedy this problem a sampling of nodes in a manner similar to that used by the Tapestry nearest neighbor algorithm [15] is performed. Starting with the highest level l in its routing table, a node contacts the fingers at that level and retrieves their fingers or inverse fingers. Here, the highest level means the last entry in the routing table. The latency to every node is measured and all but the k closest are discarded. The node decrements l and retrieves the level-l neighbors from each non-discarded node. This process repeats until $l < 0$ [30].

Differences in Performance. Only a little of global sampling is necessary to improve latency in a great extent versus the version without PNS. Comparing with global sampling the other versions are not much more efficient. This is the result of a study from S. Rhea et. al [30].

Obviously this agrees with the contention of Gummadi et al.[28] that only a small amount of global sampling is necessary to achieve near-optimal PNS. But as explained in 4.1.2 on page 31 the probability of getting the closest node decreases the greater the examined id-range is, when sampling only 16 consecutive nodes.

Finally no version can dominate the others [30].

Improving PNS with Space Mapping The idea is to divide the two dimensional euclidean space into distinct areas of same size. Distinct latency areas are assigned to the regions. Nodes have to compute their Vivaldi coordinates first to detect their region. A node which is placed in a region finds its latency-closest neighbors in the same region. Further, an id is assigned to each region.

A node's id is randomly produced like in the other DHT-based overlay networks.

The way a node gets knowledge about its latency-nearby nodes is the following: A node must be aware of a certain node, a so-called cluster node. To find the CN[3] the new node hashes the region's id and performs a lookup for the resulting key. The node which is the latency-nearest to the key is the CN.

This CN is responsible to inform the nodes about the arrival of new nodes in the region. Hence, the nodes have the possibility to find nearly all latency-nearby neighbors through the CN. This extraordinary node is the main problem: High Churn[4] can let the CNs disappear. Unfortunately no consideration of Churn was done within this study by Hancong Duan et al. [31].

[3]CN: Cluster Node

[4]A network with a lot of participating nodes faces a continuous departure and arrival of nodes. This continuous process is generally known as Churn.

4.2.3 Effectiveness of PNS and PRS

Obviously PNS is more efficient than PRS. This is deeply rooted in the fact that PRS has a more bounded choice than PNS has. To understand this characteristic one can imagine a routing process of a query. The query's target is at distance $[2^i, 2^{i+1}]$ away from the first node itself.

With PNS the node selects the i-th neighbor in its routing table. The selected finger for this entry has been selected out of 2^i neighbors from the id-distance interval $[2^i, 2^{i+1}]$.

With PRS the node performs a selection for the next hop only out of its first i fingers. In contrast to PNS, all of these fingers have been selected deterministically without previous latency consideration.

Therefore PNS is much more improving the latency than PRS does. One should be aware, this fact belongs to $IdealPNS$ [28].

4.3 Implications of PNS and PRS

This section chases the impact of PNS and PRS on lookup latency and the resilience of DHT-based overlay networks, especially under a high degree of node arrivals and departures. It is tried to answer the question if PNS or PRS produces problems when considering the mentioned issues.

4.3.1 Churn and PNS/PRS

A network with a lot of participating nodes faces a continuous departure and arrival of nodes. This continuous process is generally known as Churn. The session time as metric can be used to measure Churn. The session time describes the time between a node joins the network until it leaves. Short session times represent high Churn rates and therefore high growth in latency.

Also heavy background traffic and network realities like queuing, cross traffic and the loss of packets can lead to difficult handling even in lower Churn rates.

All the tested DHT implementations which have been examined by S. Rhea et al. [30] showed inconsistencies like failing lookup requests and inconsistent results from different source nodes under high Churn:.

Carefully chosen timeouts and bandwith usage are possible ways to face this problem. Also PNS and PRS are ways to minimize the resulting high latency rates caused by Churn. All this can help to minimize the failure rates of lookups and to decrease lookup latency.

Considering Churn the time allowed to be elapsed before reacting on changing network conditions[5] should be carefully chosen .

Under a low level of Churn a reactive recovery is highly efficient and the periodic one is wasteful. But if the level of Churn rises, the reactive recovery shows more drawbacks. Compared to the periodic proposal it becomes very expensive. The periodic proposal aggregates all changes and adapts at once whereas the reactive does consider only a part of changes at once [30].

High Churn is able to cause a significant increasing lookup latency. Up to now there has been no consideration of how to handle timeouts. The timeout value a DHT chooses can greatly affect the performance under Churn. When a query is forwarded to a node which is no longer part of the network, the requesting node waits a certain time and in the case of no acknowledgment the request is tried again, maybe with another node. Obviously, this processing leads to an increased query latency. Hence, it is great challenge to select the right timeout. With regard to high Churn rates the careful calculation of timeouts can be more worth than PNS. As explained they can increase the latency to great extent. With it timeouts are very critical to lookup latency in the time of high Churn rates [30],[32].

Hence, in the face of high Churn rates, it is much more important to possess fingers which progress the query rather than a latency closest nodes: An immediate disappearing latency close node is useless.

4.3.2 Proposals to avoid High Latency under Churn

1. **Sequential Fingers** . Side-stepping fingers with similar properties like low latency and the nearest id to the origin finger are a most suitable solution. So a sufficient number of these sequential neighbors reduces the Churn issue like Yingwu Zhu et al. [32] explain.

   Therefore no reactive recovery is really useful. Obviously the mentioned additional effort for a latency-closest finger which disappears immediately cannot be justified. Additionally this version can lead to positive feedback cycles as Yingwu Zhu et al. [32] explain. So periodic recovery is the more suitable solution and a lot of changes can be worked up at once.

2. **Liveness Node Selection**. To avoid latency increase caused of timeouts a different neighbor selection can be performed. Fingers should be selected with regard to their liveness, i.e., if they are long-living they illustrate high quality fingers. This version is called Liveness Node Selection. With it the recovery process can be done within greater time intervals. The competition of user lookups and recovery messages is lowered, too. LNS[6] can degenerate to PNS if each node offers the same liveness [32].

[5] Usually this is known as timeout.
[6] LNS: Liveness Node Selection

3. **Biased Route Selection**. A combined consideration of LNS and latency is BRS[7]: It is tried to avoid increasing latencies caused by timeout failures. This new proposal has been made by Yingwu Zhu et al. [32]. The decision over the next hop is made of neighbor's liveness probability and latency characteristic.

   This brings up two advantages:

   (a) No latency increase because of timeout failures.

   (b) Nearby neighbors: Timeouts are quick.

   That means the following: Low latency links without the cost of increased hops through neighbor failures.

4. **PNS: Avoiding local Minimums**. Imagine a low Churn rate, there is a chance PNS can stabilize at some entry which is on a local minimum. If an entry remains optimal after considering the known candidates k times one can turn to another PNS-technique, e.g., inspired by global tuning. The hope is to sample a set of nodes outside the optimized entry's candidate set [24].

4.3.3 Unbalanced Overlay Structure: High in-degree Nodes

Neighbor selection policies can produce unbalanced overlay structures. When only some nodes are linked by a great number of other nodes the overlay becomes vulnerable for attacks.

So it is imaginable that these high in-degree Nodes present a higher risk for the overlays resilience. When these nodes disappear a lot of important links can fail and the network becomes instable. To reduce this issue a sufficient number of sequential neighbors can be maintained [32]. Sequential neighbors represent additional fingers of one id-range.

After Yingwu Zhu et al. [32] the combination of PNS and a sufficient number of additional fingers can strike a balance between attack resilience and lookup performance.

4.4 PNS and PRS in Chord

This section gives an outline over PNS and PRS algorithm implemented in the known DHT Chord [8]. Constraints in implementing PNS or PRS are presented. The advantages and the drawbacks are shown.

Chord [8] is another DHT implementation. In contrast to IGOR, Chord uses an asymmetrical metric. In the following Chord's operation mode is shortly outlined.

[7] BRS: Biased Route Selection

Chord: Overview [8]. Like IGOR, Chord's geometric interpretation illustrates a ring structure like it is shown in Figure 2.2 on page 18. But, unlike IGOR, Chord implements an asymmetric metric. So the node responsible for a key is the node which successes the key directly. This node is called the key's successor.

All keys which are numerically placed between two nodes belong to only one node: The direct key's successor. Chord uses a finger list to make routing efficient. The fingers in this routing table are ordered with exponentially growing id-distance to the Chord node. This results in an average query hop-count of $O(log\ N)$, where N is the number of participating nodes. The routing consists of forwarding the query to the finger which is numerically closest, similar to IGOR.

Chord and PNS. The ring structure Chord implements offers a degree of flexibility in selecting fingers for the finger table. Each finger can be selected from a distinct portion of id space. The sizes of these address portions illustrates the degree of freedom in selecting neighbors. Every node which refers to such an id-distance interval is selectable as a wanted finger.

In more detail, the entry for the $i-th$ finger of a node with id a in the finger table is chosen from the interval $[a+2^i, a+2^{i+1}]$. One can conclude a range of 2^i possible fingers for the $i-th$ entry.

The crucial question is how to obtain these possible fingers, i.e., how to obtain knowledge of them. Therefore Global Sampling as described in 1 on page 33 can be used. Global Sampling tries to find the mentioned neighbors. So lookups for random keys with appropriate prefixes are made. Latency is directly measured or predicted with Vivaldi which was introduced in 3.1 on page 25 and the corresponding finger entry is replaced or not.

To overcome Churn a list of sequential fingers is maintained. This procedure is introduced in 1 on page 37

Chord and PRS. Imagine two nodes which are $O(N)$ away from each other, where N is the number of participating nodes. The first can choose from $log\ N$ fingers which are able progress the query. The next node can choose from $(log\ N)-1$ fingers. And so on. All in all an average hop-count of $O(log\ N)$ is achieved, where N is the number of participating nodes. So the PRS algorithm chooses always the node which grants progress in id-space and which offers the best latency cost.

Results. The analysis of Gummadi et. al [28] shows lookup latency can be significantly improved with PNS. Over 50 percent of the lookup are within a latency of approximately 120 milliseconds at maximum. In contrast without PNS approximately 900 milliseconds at maximum.

PRS improves the latency only at the half of the above mentioned milliseconds.

Combined with sequential fingers the static resilience is improved.

4.5 PNS-Proposals suitable for IGOR

This sections examines which of the known PNS-proposals is suitable for the IGOR overlay network. Firstly the flexibility in selecting fingers is examined.

4.5.1 Degree of Flexibility in Selecting Fingers (Neighbors)

As mentioned above and showed in earlier sections IGOR implements a virtual ring structure like Chord. With it degrees of freedom in selecting fingers are given. In more detail, IGOR tries to built a finger table with entries which id-distances grow exponentially in id space to the IGOR node itself. The distance between the fingers stays logarithmically equal. Especially, it is tried to achieve a logarithmic dualis id-distance of 1 between successive fingers as shown in Figure 4.3 on page 42.

One can say, this is nearly the same proposal as implemented in Chord. But in contrast to Chord different ids for each finger entry are possible. Mainly this is caused by the symmetrical metric. Not only the clockwise ids in the mentioned id-distance can be examined for new entries, but also the anticlockwise ids can be new candidates. This is shown in Figure 4.2 on page 41.

Obviously if the id-distance to the node increases the number of possible fingers increases, too. In other words flexibility accumulates the more id-distance is considered.

4.5.2 Obtaining the Neighbors with the Appropriate Identifiers

Within this section the suitability of the proposals like those mentioned in 4.2.2 on page 33 are examined regarding IGOR.

1. **Global Sampling and IGOR: Suitability and Expectations**. This is the most preferred one because of its simplicity. New nodes can be found through lookup queries for keys in the appropriate id-ranges. The nodes which are responsible for those ids answer and can be selected. Regarding IGOR, it is necessary to lookup for keys in both directions.

   Therefore a lookup for a certain id-distance can be made.

   Hence, Global Sampling is suitable. But it comes with the same insufficiency as mentioned in 1 on page 33.

2. **Finger's Fingers: Suitability and Expectations**. It is imaginable that a finger also holds latency-nearby fingers with suitable ids. But the requested finger tries to update its finger table with latency-close nodes, too. Therefore it can be expected that the nearest finger holds also latency-close fingers for the requesting node.

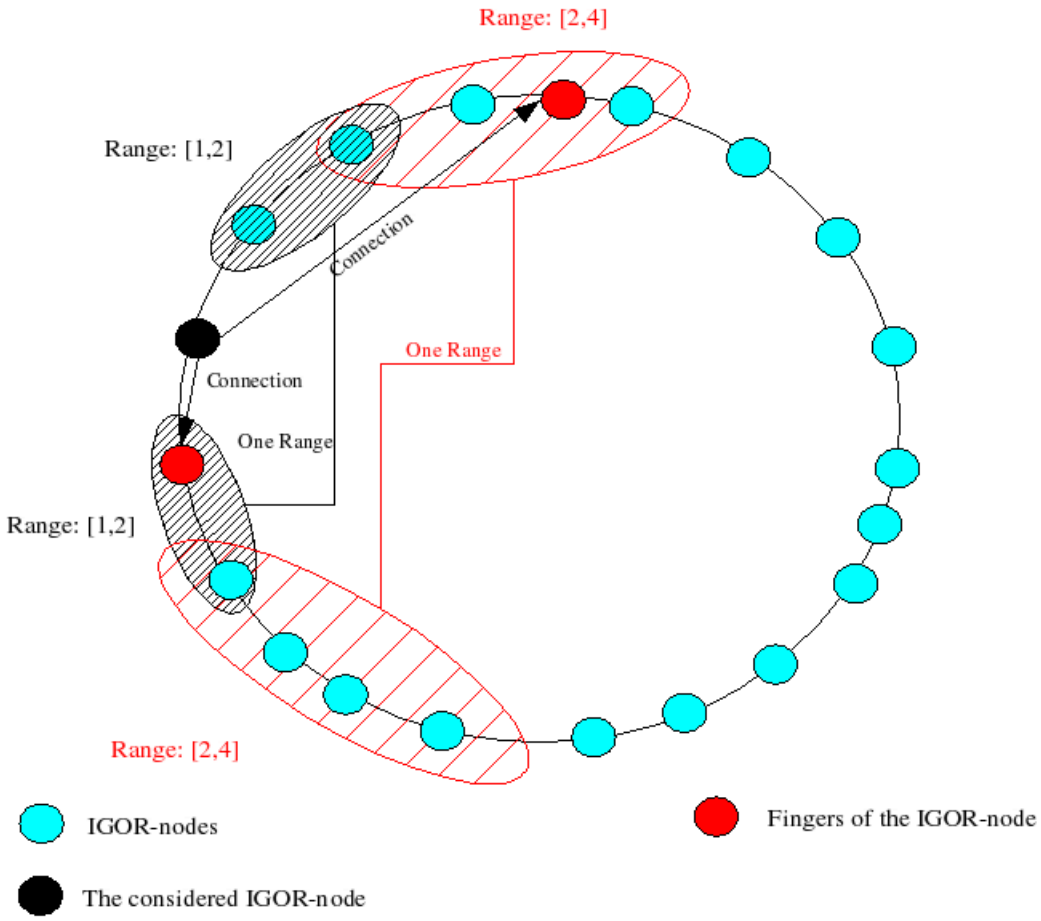

Figure 4.2: As illustration only two ranges are shown: Symmetric finger ranges $[2^0, 2^1]$, $[2^1, 2^2]$ for the considered "black" node. The arrows point to connected fingers. Easy to recognize are the "doubled" ranges. The areas which belong to each other are similar hatched and colored.

3. **Finger's inverse Fingers: Suitability and Expectations**. Like the proposals before, this one is suitable, too.
4. **Recursive Sampling: Suitability and Expectations**. Like the proposals before, this is suitable, too.
5. **Space Mapping: Suitability and Expectations**. This is imaginable, but the problem is the mentioned cluster node under CHURN. For more clearness see 4.2.2 on page 35.

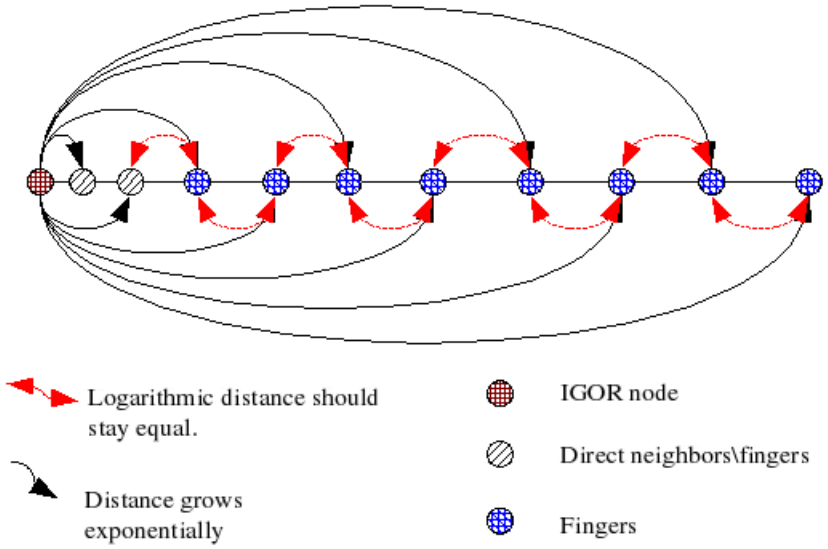

Figure 4.3: The finger list in IGOR: It is tried to obtain an uniform distribution of a node's fingers. The id-distance between fingers should grow with factor 2. The id-distance of the fingers to the node grows exponentially with base 2.

4.6 Proximity Route Selection in IGOR

IGOR is able to implement PRS. The next hop is selected with regard to its latency characteristic and its id-closeness to the target. Especially, the fingers which can be used as a next hop are examined by their latency, and the best one is selected.

The PRS can also be used after PNS has been performed to improve the overall lookup latency again.

Chapter 5

Merivaldi: A PNS-Approach for IGOR

The basic idea is to combine the Meridian [4] Closest-Node-Discovery with the advantages of synthetic coordinates to obtain latency-proximate nodes for the finger table. Within this chapter the basic Meridian procedure is described. Merivaldi a promising proposal of PNS is introduced and analyzed.

5.1 Meridian

Meridian is a framework for node selection based on network location properties. It offers a multihop search for a closest node where each hop reduces the distance to the target. It always finds the closest node exactly or near exact [4]. The framework ensures that the next hop on the way to the closest will be closer or at least equal.

5.1.1 Basic Framework

The basic Meridian framework consists of three parts: First, a ring-structure each Meridian node must manage and which is important for the search algorithm. Second, a ring membership management which actualizes and improves the ring characteristics. Third, a gossip-based node discovery protocol to obtain new candidates for the rings.

1. **Multiresolution Rings**. Each Meridian node maintains $k = m \times O(log\ N)$ nodes, called ring-members to resolve queries in $O(log\ N)$, where m is the number of rings and N is the number of nodes participating in Meridian. Simulations show that a small k suffices [4]. With regard to their latency, these ring-members are organized

into concentric non-overlapping latency-distance rings around the Meridian-node itself. The rings' size grows exponentially with growing latency-distance. A node/ring-member belongs to a ring through its latency. The count of rings is finite and the number of ring-members each ring harbors is limited. A ring is described with two circles of different radii. The i-th ring has an inner radius $r_i = \alpha s^{i-1}$ and an outer radius $R_i = \alpha s^i$ for $i > 0$, where α is a constant, s is the multiplicative increase factor, and $r_0 = 0, R_0 = \alpha$ for the innermost ring. All rings $i > i^*$ for a system-wide constant i^* are aggregated into a single, outermost ring that spans the range $[\alpha s^{i^*}, \infty]$ [4]. This sorting of neighboring nodes is performed by any Meridian node.

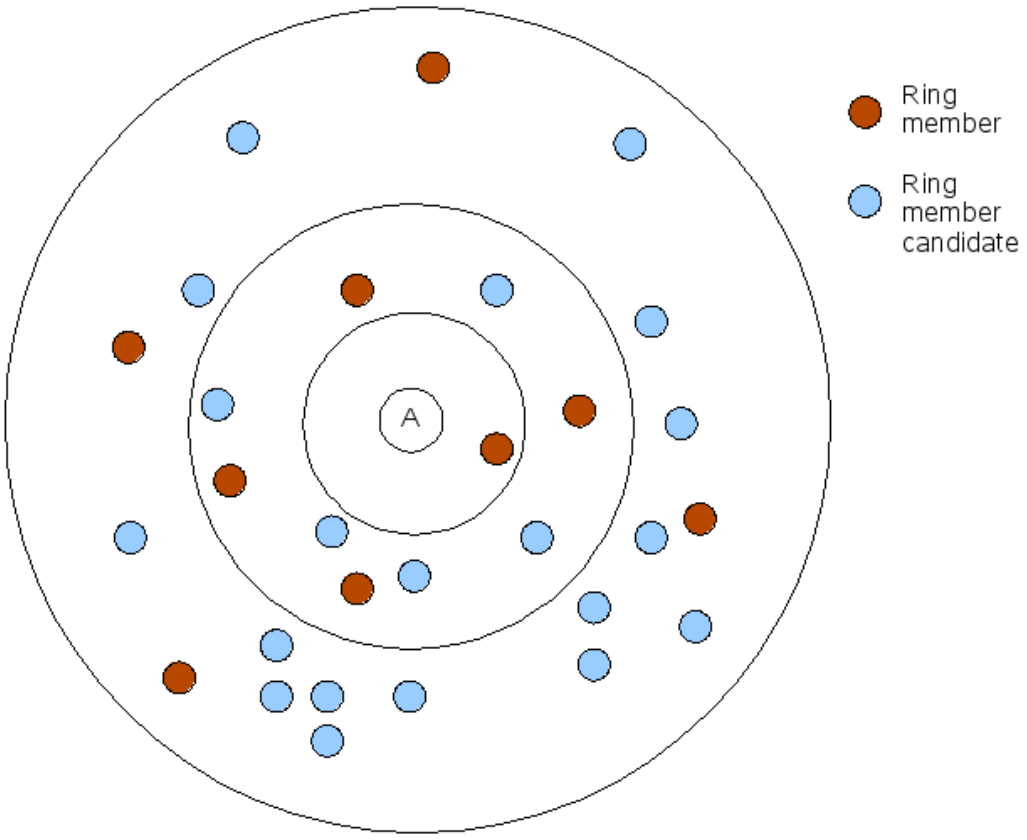

Figure 5.1: The Meridian rings. The red colored nodes are selected to be ring-members. They offer the greatest geographic diversity. The blue colored ones are ring-member candidates which are kept in mind to overcome Churn. The rings itself illustrate exponentially growing latency-distances.

2. **Ring Membership Management**. The number of ring-members per ring is a tradeoff between accuracy and overhead. Large numbers increase the overhead of maintenance. Small numbers decrease the accuracy of the impact of built-on algorithms like the Closest-Node-Discovery. The Closest-Node-Discovery is introduced in 5.1.2 on page 45. So the number of ring-members each ring harbors is limited

to a constant k. Anyhow a Meridian node keeps track of secondary ring-member candidates to overcome Churn problems.

The selected ring-members within a given ring can be significantly responsible for the performance of the Closest-Node-Discovery. Hence, a set of geographically distributed nodes per ring provides better progress in finding the closest node than rings with geographically clustered nodes. The reason for that is precised later in 5.1.2 on page 45.

Therefore a ring-member will periodically measure its distance to the ring-members of its own ring. Each measurement is stored and sent to the Meridian-node. After the Meridian-node has received each latency measurement of its ring-members the Meridian-node determines the maximal hypervolume polygon[1] per ring. For a small number of possible ring-members it is possible to determine this polygon by considering each possibility. For a big number, a "greedy algorithm"[2] takes place [4].

3. **Gossip based Node Discovery**. This protocol ensures new ring-members can be found and selected. It is necessary to obtain new possible ring-members when Churn is high to provide a good ring-membership quality [4].
   (a) Each node A randomly picks a node B from each of its rings and sends a gossip packet to B. The gossip packet contains one randomly selected ring-member per ring.
   (b) When a packet is received, node B determines through direct RTT-measurements its latency to A and to each of the ring-members contained in the packet. Then it places the potential ring-members in its rings or stores them as secondary candidates.
   (c) A waits until the next gossip period is to be performed.

5.1.2 Closest-Node-Discovery

Generally a multihop search where each hop reduces the distance to the target is performed.

A Meridian node which wants to get knowledge about the closest node to a certain target node, determines the RTT d between itself and the target through directly measuring the RTT. Now the Meridian node queries all of its ring-members whose latency distances are within $(1 - \beta) * d$ and $(1 + \beta) * d$.

The queried ring-members now measure their distance to the target and report the results back to the requesting node. This processing is illustrated in Figure 5.2 on page 46. The

[1] A polygon is a plane figure that is bounded by a closed path, composed of a finite number of sequential line segments.

[2] A greedy algorithm always takes the best next solution without considering further steps.

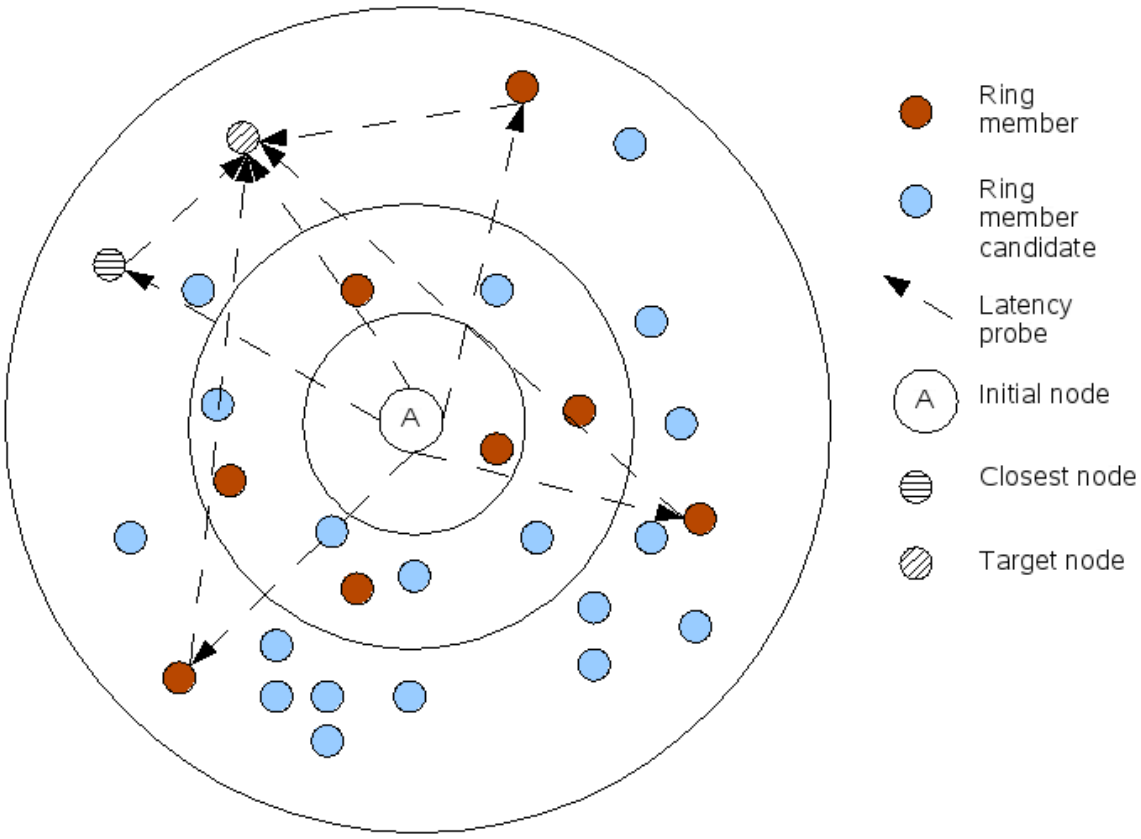

Figure 5.2: Closest-Node-Discovery. Node A wants to get the closest node to the target. The Meridian-node measures its own RTT to the target. All ring-members in the appropriate latency-ring are requested to measure their RTTs to the target. This is illustrated by the arrows. The requested ring-members answer with their RTT to the target. This process continues until no closer node is found.

ones which take more time than $(2\beta + 1) * d$ are out of further consideration. They cannot be closer [4]. Then the Meridian-node determines if there are ring-members closer than it is. If there are closer ring-members the query is forwarded to the closest one.

5.2 Merivaldi: Vivaldi combined with Meridian adapted to IGOR

The basic idea is to perform the latency measurements done by Meridian through latency prediction derived from Vivaldi-coordinates. Using latency prediction provides a great positive effect: One does not have time loss through direct RTT-measurements. No prior communication is necessary. The combination with Meridian offers another advantage: A

closest node can be found within $O(log\ N)$ hops where N is the number of participating nodes. To gain not only one closest node, initially different nodes are requested to find closer nodes. The target is the Merivaldi-node. The following conversion of this idea is done in consideration of the IGOR overlay network. Firstly, the Merivaldi framework which consists of a ring structure, a ring management and a gossip protocol, is introduced, then the basic Merivaldi algorithm is explained.

5.2.1 Ring Structure and Management

Ring Structure. The rings will be filled with arbitrary ring-members first: Joining the overlay network, a node will know its first ring-members through the TCP-connections it is establishing. These first ring-members/fingers are placed in the rings by means of their latency to the Merivaldi-node. After that the ring management takes place.

Ring Management. The ring management addresses two important aspects: On the one hand being up-to-date when Churn arises. On the other hand it decides which ring-member is allowed to stay in its ring: The so-called ring membership.

1. The most ring-members are node-references to which no direct TCP-connections exists. A node-reference presents a set of a node-id, transport-address[3] and Vivaldi coordinates. The liveness of the ring-members must be indictable because this is not obvious.
   The solution is: To reduce the probability of having"nodeless" identifiers in the rings a time-value is added to any ring-member. This value contains the time the inserted ring-members are known and is used as a liveness indication. Obviously, only the node whose fingers are ring-members, too, is able to start and to refresh this time-value by zero: The ring-members are its own fingers to which it holds direct TCP-connections. If such a ring-member is published over the overlay to the next hop this one is able to determine about its liveness. (The delay between two hops can be computed by their coordinates and this delay can be added up to this time-value, too. Between receiving and sending time goes by and can be added up additionally. Merivaldi does not regard this because the additional overhead and the questionable advantage.)

   On the assumption that only "short" living periods of nodes exist it is simple to say node-identifiers with a high time-value can be offline to a greater probability than ones with low time-values. So one will have the ability to constitute which ring-member should be out of consideration. This information is important for the next step of improving the rings' memberships.

2. The number of ring-members per ring must be limited: The refreshing, i.e., updating the ring-members which is explained later in Ring Member Selection means overhead. The solution is: An important condition is that ring-members which are also own

[3] A transport-address consists of an IP-address and a portnumber

fingers, should stay in the rings as long as they exist. Only these can be refreshed to a time-value of zero as explained above. This is necessary for the refreshment of the time-values. This is later explained in "Merivaldi Gossip Protocol".

The deletion of a ring-member depends on its age and the overpopulation of its appropriate ring: If a ring is overpopulated, the ring-members are sorted by their age and the oldest ones are eliminated. On the basis of this processing each ring-member which is known through the later explained Gossip Protocol in 5.2.1 on page 5.2.1 is eliminated if it is not refreshed. This also provides a small positive effect regarding security: "Bad ring-members" are eliminated after a certain time if they are not refreshed.

Ring Member Selection. If a new potential ring-member is placed in the appropriate ring of a Merivaldi-node, depends on its age, the number of already existing ring-members and the determined latency to the Merivaldi node. The following describes the algorithm to select the appropriate ring-members.

1. Iterate through the given ring-members and place them in the corresponding rings through determination of their latency to the Merivaldi-node. Ring-members which are own fingers are placed there before. This was explained in Ring Member Selection 5.2.1 on page 47.
2. Determine the relative online time of each ring-member and sort the ring-members by age per ring. Eliminate the oldest ring-members which are overpopulating the ring.

Merivaldi Gossip Protocol The Merivaldi-node selects a random ring-member per ring. It sends a GP-message[4] which contains one random selected ring-member per ring to these ring-members. The receiving nodes now can place these potential new ring-members in their rings. If they are already known it can refresh the existing ones. Refreshing means updating the time-values of the already known ring-members. The higher time-value is changed to the lower one.

5.2.2 Closest Node Discovery: Basic Algorithm

The basic algorithm of the Closest-Node-Discovery is explained here.

Merivaldi performs the Closest-Node-Discovery periodically. An iterative and a recursive version of the algorithm is imaginable. In the following it is described why the iterative version is preferred.

Recursive version

[4]GP-message: GOSSIP-message

1. A Merivaldi-node periodically sends a Closest-Node-Job to an arbitrary ring-member per ring.
2. The ring-members answer with their ring-members which are latency-closer to the Merivaldi-node than they are.
3. The Merivaldi-node computes the latency prediction to the received ring members. It is able to determine the latency because each ring-member contains a Vivaldi-coordinate.
4. The Merivaldi-node determines if closer ring-members in the ring-members' answers exists than the requested ones are. If there is one, the closest node request is forwarded there. This is done until no closer node can be found.

Iterative. It is also possible to let the requested ring-member per ring determine the latency predictions to the requesting Merivaldi-node. Then the ring-members are responsible for the forwarding and the answer for the Merivaldi-node. In this version the CND-job [5] is forwarded by the ring-members until no better node is found. The last node on the way reports itself to the Merivaldi-node. The iterative version is preferred because not every ring-member needs to answer the Merivaldi node. In contrast to the recursive version, this results in a decreased network burden. Additionally the Merivaldi-node does not have to determine the closest ring-member. Hence, the determination of the closest node is more distributed. For more see 6 on page 57.

5.2.3 Vivaldi: Model Selection, Probing and Positioning

The best space-model with the lowest error must be determined for Merivaldi. It is important to keep the prediction error low as far as possible. With these latency predictions decisions about closer nodes will be made. As earlier mentioned in 3.1 on page 25 three important aspects have to be considered: Space selection, probing and positioning.

- **Space selection** means how to calculate the distance between two points. Possible spaces are euclidean, spherical etc. Here it is the N-dimensional euclidean space which can be used well as Tang et. al [33], [24] explain. Additionally it is easy to implement a possibility to modify the number N of used dimensions.

  Merivaldi implements the N-dimensional euclidean space-model. In other words it is possible to use an arbitrary N-dimensional euclidean space. The more dimensions are used the more accuracy is assigned. One will have more independent directions to move for hard-to-place nodes [5]. But there is to say that more dimension mean more overhead, too. Because the 4-dimensions outperform merely the height-vector model with two-dimensional euclidean coordinates, Merivaldi is able to use any N-dimension. Unfortunately the height-vector-model is only analyzed with two

[5] CND: Closest Node Discovery

euclidean coordinates. More reasons for not using the height-model have been explained in 3.1 on page 25. But it would be easy to adapt Merivaldi to new cognitions.

- **Probing** means the process to get the actual RTTs between two nodes. An IGOR-node uses periodically sent PING-messages to keep TCP-connections alive. A node obtains these PING-messages from all of its fingers. Hence, the PING-messages is used to piggy-back the needed information for updating coordinates. More explanation why the PING-message is used is provided in 6.2.2 on page 59.
- **Positioning** means using the probe results to assign location in the chosen space. Here the Vivaldi algorithm as described in the Vivaldi section 3.1 on page 25 will be used.

Each node must maintain synthetic coordinates obtained from the Vivaldi algorithm. Only when nodes are extended with Vivaldi coordinates they are able to predict latency. One can consider the coordinates should have converged to their "final" position before the periodically Closest-Node-Discovery provides good results. Despite this reason a CND-job is performed. The more time goes by the more accurate the results will be. It is not possible to determine if a coordinate has reached its final destination. All coordinates are changing their values dynamically all the time. A hint is a low relative prediction error. One can decide to perform CND-jobs only when it is low. This only presents a questionable advantage when a fresh node joins the overlay network. Hence, Merivaldi will not regard this.

5.2.4 Analysis of Merivaldi

In this section Merivaldi is examined analytically. The number of messages, different message types and the size of a message is analyzed. Another interesting statement is how many hops a CND-job needs on average. Additionally the complexity of the ring management is considered. It is shown that Merivaldi comes with a modest network burden. The costs for the ring management in the worst case are within $N \times O((M+1)log(M+1))$ where N is the number of rings and M is the number of ring-members.

1. **Message Overhead Calculation**. A distinction of the Merivaldi system parts is made to get knowledge about the fact where overhead is produced. There are three main parts of Merivaldi: Vivaldi which assigns coordinates, the ring management to update the ring-members, and the CND-job.

   For the calculations the following is presumed:

   - An **identifier** contains a size of 30 characters as one can see here, for instance, 8a93090bc2159d9d830195dd4cd92b. Therefore $30 \times 8bit = 0.03kbyte$ per identifier.

- A **node-reference** represents a node-id followed by its IP-address and portnumber, and its 5-dimensional coordinate plus remote error. It has a size of 107 characters as one can see here, for instance, (6d70a47b7a8a93090bc2159d9d830195dd4cd92b 127.0.0.2:29434 +11050.7,8769.25,1628.34,5537.95,8698.17# 0.58846). Therefore $107 \times 8bit = 0.107kbyte$ per node reference.
- A **time-value** consists of a double value. Therefore $64bit$ each.
- "**Additional data**" is defined as the basic amount of data, which each message-type of IGOR exhibits.

The calculations:

- **Vivaldi**. No additional messages are necessary. The needed information is piggy-backed by the PING-message-type. For more detail see 6.2.2 on page 59

  This means additional 6 double-values, one per coordinate and one for the remote error. Because these values are converted in a string like "—0.01234,1.02434,2.02432,3.34220,4.23426,5.0# 0.43435)" with 46 characters one have $51 \times 8bit = 0.051kbyte$ additional data per PING-message. Note, a PING-message is periodically sent each n time units.

- **Gossip Protocol**. A Merivaldi-node must know a lot of ring-members as explained in 5.2.1 on page 47. Merivaldi gets to know them by updating the rings with the GP-protocol defined in 5.2.1 on page 48. The overhead one node produces, strongly depends on the number of rings. If a node maintains m rings, at most m GP-messages will be sent per GP-period. Each of these with at most $m-1$ ring-members as content. Additionally it is important to refresh the ring-members. Therefore a time-value is added to a ring-member as in 5.2.1 on page 47 introduced. That means, at most $m\times$ GP-messages with at most $m-1$ ring-members where each consist of a node reference and a time-value , i.e., 107 characters per node-reference, hence, $0.107kbyte$, and the time-value which is represented as a double-value and some additional data. This is

$$\begin{aligned} m \times (m-1) \times (0.107kbyte + 0.008kbyte + additional\ data) = \\ = m^2 - m \times (0.115kbyte + additional\ data) \end{aligned} \tag{5.1}$$

  per gossip period. For instance, if Merivaldi uses 9 rings and 5 ring-members each, with it the network burden is

$$72 \times (0.115kbyte + additional\ data) = 72 \times additional\ data + 8.280kbyte \tag{5.2}$$

  for a gossip period per node. To add is that the GP-messages are sent periodically.

- **Closest Node Discovery**. A Closest-Node-Job-message is sent to 1 ring-member per populated ring. Then the CND-message[6] is forwarded by the IGOR-overlay-participating nodes. This is performed until a closest node for the requester is found. This one is reported by another message type, called Closest-Node-Job-Answer-message. A CND-message contains the requestor's node-reference, a tracking of already visited node-ids and some additional data like any other message.

$$\begin{aligned} m \times (0.107kbyte) + m \times (0.03)(\ for\ the\ first\ tracked\ identifier) + \\ + m \times additional\ data = \\ = m \times additional\ data + m \times 0.110kbyte. \end{aligned} \tag{5.3}$$

Assuming 9 ring-members each ring, the network burden for the first CND-hop[7] is

$$9 \times additional\ data + 0.990kbyte. \tag{5.4}$$

The CNDA-message[8] will not be routed back by all last CND-receivers: Only if no appropriate connection exists. For this computation is assumed each CND-message is answered. It contains a new finger suggestion, i.e., assuming 9 sent CND-jobs the burden is

$$9 \times (0.107kbyte + additional\ data) = additional\ data + 0,963kbyte \tag{5.5}$$

All in all this is for the Closest-Node-Discovery of a Merivaldi-node without intermediaries:

$$2m \times additional\ data + m \times 1.1kbyte \tag{5.6}$$

For instance, assuming n intermediate hops each, the message size grows caused by the tracking of visited nodes:

$$n + n - 1 + n - 2 + ... + n - (n + 1) = \frac{n(n+1)}{2} \times 0.03kbyte \tag{5.7}$$

additional burden for the network.

Each CND-hop decreases the latency-distance like it is explained in Meridian [4]. One has an average CND-hop-count of $O(log\ N)$, where N is the number of participating nodes.

To avoid loops, which can occur when network conditions change, a forwarded CND-message tracks node-ids until a maximum of 30. If this threshold is reached the CND-job is finished.

[6] CND-message: Closest-Node-Job-message
[7] CND-hop: Closest-Node-Job hop
[8] CNDA-message: Closest-Node-Job-Answer-message

An example for a summary-calculation of the network burden.

For this calculation the following is presumed:

- A GP-period[9] is done every 30 seconds.
- A CND-period is done every 20 seconds.
- Additional data is not considered, but must be not forgotten.
- The network burden is considered for 1 hour.
- 9 rings are implemented.
- Each CND-job needs 10 hops.
- 1000 permant IGOR-nodes.
- Vivaldi piggy-backed messages are sent every 20 seconds to all known fingers. It is assumed that a node holds 10 fingers.

The network burden can be estimated by the following computation:

$$\begin{aligned} 1000 \times (60 \times 2 \times (8.28kbyte) + 60 \times 3 \times (9 \times ((0.107kbyte + 0.107kbyte) + \\ + \frac{10 \times (10+1)}{2} \times 0.03kbyte)) + 10 \times 0.051kbyte) = \\ = 1000 \times (993.6kbyte/h + 348.33kbyte/h + 91.8kbyte/h) = \\ = 1433730kbyte/h = \\ = 1.438Gbyte/h \end{aligned} \tag{5.8}$$

This means an average network burden of $398.258kbytes\ per\ second$ for the Merivaldi-algorithm on 1000 nodes and per node $0.398kbytes\ per\ second$ on average. In times of high-speed internet this is only a little burden.

2. **Ring Management**. The ring management takes place when a GP-message is received. Usually a Merivaldi-node harbors $O(log\ N)$ ring-members per ring, where N is the number of participating nodes. Therefore one Merivaldi-node has $m\ O(log\ N)$ ring-members, where m is the number of rings and N is the number of participating nodes. The number of GP-messages a Merivaldi-node sends is m per GP-period. A known ring-member is selected as a GP-target by the probability of $\frac{m}{m\ O(log\ N)}$, where N is the number of partcipating nodes and m is the number of rings. This must be multiplied with the probability k of being a known ring-member per Merivaldi-node. Therefore a Merivaldi-node receives $\frac{k}{O(log\ N)}$ GP-messages per period.

   A GP-message contains in the implementation of Merivaldi in IGOR n potential new ring-members at maximum, if $n+1 = d$ rings exist . A ring is fully populated when m ring-members are inserted. If a new potential ring-member is received, it is necessary

[9] GP-period: Gossip period

to find out if this one is already a ring-member. If it is already a ring-member, its time-value, the relative online time is updated with the newest time of both. After that the ring-members are sorted by their age and in the case of overpopulation the oldest ones are deleted.

For the following calculation only one received GP-message is considered: Assuming each ring is full populated with m members and a GP-message contains n potential new ring-members. It is $n < \#rings$ because each GP-message can only contain $\#rings - 1$ ring-members as explained on 5.2.1 page 48. It can be the worst case that each of these new ones are distributed in a way that each new member belongs to a different ring. It is assumed that Quicksort[10] is used for sorting. Here we have maximum sorting costs of $n \times O((m+1)log(m+1))$. is used. If all x new ring-members belong to the same fully populated ring, the sorting costs are minimized to $O((m+x)log(m+x))$.

To determine a Merivaldi-node's whole costs for for one GP-period one can multiply this sorting costs with the number of GP-messages $\frac{k}{O(log\ N)}$, where N is the number of participating nodes and k is the probability of being a known ring-member per Merivaldi-node.

3. **Iterative vs. Recursive Merivaldi**. The iterative version distributes the computing of the best node better and avoids an increasing message-appearance as mentioned in 5.2.2 on page 48. But within the iterative version the CND-requester is committed to an answer of a node which is not known before. Note, it is not necessary to get an answer for the further operation of Igor.

4. **Getting the Closest Node**. The new found node will be nearer or equally near than the old one. This is obvious because of the forwarding of CND-message only if a closer node is found. Each node performs quality ring management. In assumption of no Churn, after a certain time the new found fingers will rest as themselves and no closer node as a new entry will be found. Additonally one can assume that the ring-members which represent fingers too, converge to the latency closest rings: Imagining the finger tables entries finger are replaced by latency closer ones. These ones are ring-members, too. Therefore these will be placed in the inner rings.

5. **Accuracy**. Vivaldi introduces errors in the prediction of latency. This can lead to non-optimal choices of new fingers. In more detail, the prediction error is responsible for the forwarding and selection of possible new fingers. The more accurate the coordinates are the more accurate the prediction and selection of the correct fingers is.

6. **Needed Connections**. No additional connections are necessary. Merivaldi works on established TCP-connections.

[10] Quicksort is a sorting-algorithm with complexity of $n\ O(log\ n)$, where n is the number of elements.

7. **Comparison Merivaldi-Vivaldi**. Merivaldi uses Vivaldi to obtain latency prediction. This latency prediction is used for the closest node operation.
8. **Comparison Merivaldi-Meridian**. Meridian is a framework which offers a Closest-Node-Discovery mechanism. It uses no synthetic coordinates. Merivaldi is a combination of both. The direct RTT-measurement of Meridian is exchanged with the latency prediction through Vivaldi.
9. **Vivaldi: Drawbacks and Advantages**. The biggest advantage is the possibility to predict Internet latency between peers which do not know each other before. Vivaldi does not need additional messages or TCP-connections to be established. Only accurate coordinates are necessary. Vivaldi is quick in adapting new characteristics of Internet latency. So it is Churn resistant. In addition, Vivaldi is totally distributed. The biggest drawback is the difficulty to map Internet latency distribution into space-models. Sometimes the Internet violates the triangle inequality and therefore the coordinates are never exact.
10. **Meridian: Drawbacks and Advantages**. Meridian offers a Closest-Node-Discovery. The closest node is found with high propability in $O(log\ N)$ Meridian hops, where N is the number of nodes participating in Meridian. The drawback is the maintaining of the rings. It is necessary to measure RTT directly.
11. **Comparison Merivaldi-Vivaldi**. Merivaldi uses Vivaldi to obtain latency prediction. This latency prediction is used for the Closest-Node-Discovery of Merivaldi.
12. **Comparison Merivaldi-Meridian**. Meridian is a framework which offers a Closest-Node-Discovery algorithm. It uses no synthetic coordinates. Merivaldi is a combination of both. The measurement of latencies done by Meridian are exchanged with the latency prediction derived from Vivaldi.
13. **Conclusion of Merivaldi**. The positive aspects of the two promising ideas are combined. Vivaldi's fully distribution and low prediction error and Meridians Closest-Node-Discovery. The drawback of Meridians direct RTT-measurements is bypassed and Merivaldi guarantees to gain closer nodes if they exist. Merivaldi decreases the latency within a CND-job for a Merivaldi node without much overhead.

Additional Consideration. Like it is described in the Meridian framework it is possible to increase the performance of the Closest-Node-Discovery with ring-members which are geographically distributed.

Then one will have the 100% propability of reducing the RTT each step. Imagine a target node to which a closer node is looked for. The RTT to this target node determines the appropriate latency-ring. Imagining the triangle inequality does not apply and the target-node is not placed in the most inner ring. Now when using a two-dimensional euclidean space and enough ring-members which are equally distributed in the ring, one ring-member will be closer to the target node. This is illustrated in Figure 5.3 on page 56.

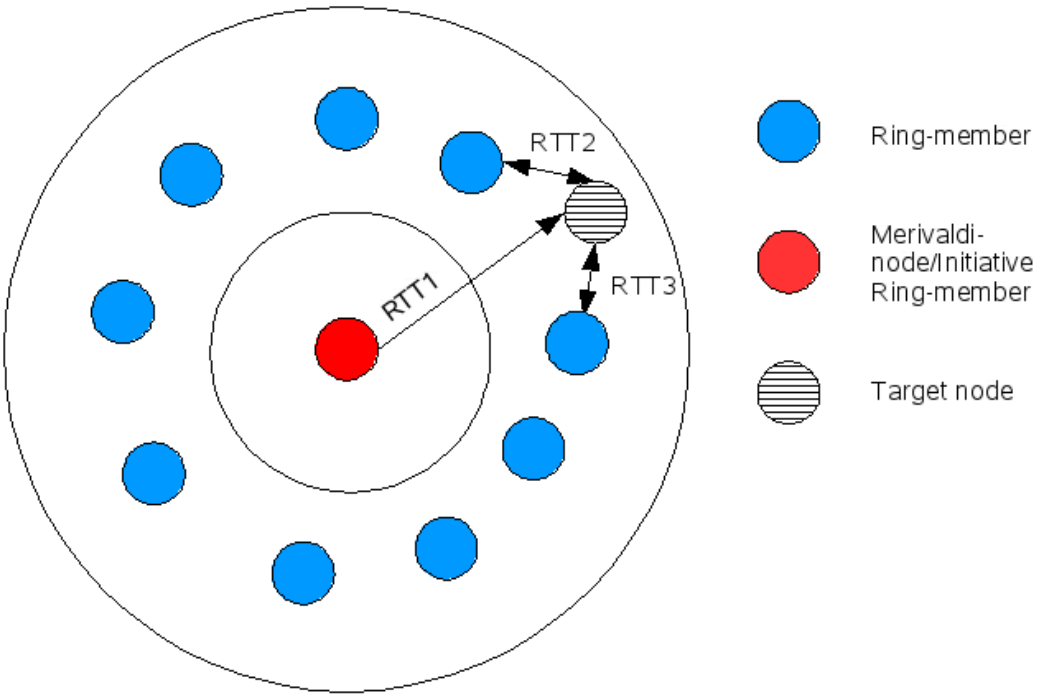

Figure 5.3: Any node with same latency to the queried ring-member will find a ring-member closer to him than the queried ring-member is. This is shown with the size of arrows. RTT1 is the RTT from the Merivaldi-node to the target. RTT2 and RTT3 show the possible candidates for the forwarding. It is easy to recognize the 100%-probability of a RTT reduction.

Unfortunately this methodology contains a lot of overhead with questionable advantage in Merivaldi. That is why it is not pursued further and not implemented, too. It can be or should be a part of further research work.

Chapter 6

Implementation of Merivaldi in IGOR

Some details of the Merivaldi implementation and Proximity Route Selection for the IGOR overlay network are described within this chapter. All in all this chapter describes how Merivaldi is adapted to IGOR.

6.1 Design Changes in IGOR

During the writing of this thesis IGOR's implementation was redefined by a more modular design. With regard to the implementation of Merivaldi this is introduced in more detail.

Basic Implementation Details of IGOR. IGOR has been tried to be designed with a modular character. Four different basic "plugin"-types are realized. To overcome confusion, the term plugin is used here in a more or less different meaning than usual. Each of these plugins is responsible for a different functionality. These are message handling (message-plugin), routing decision evaluation and connection decision evaluation (policy plugin), periodic task handling (task plugin), and a plugin which only has a consideration characteristic (peek plugin). Each of the plugins offer different functions which can be used for own purposes. A plugin here is not dynamically loadable, but added hard coded. Additional to note is that IGOR uses openend TCP-connections as finger entries.

Plugin Concept of IGOR. The plugins and their purposes are shortly explained in the following. For more detail see the free available source code of IGOR[1].

- **Policy-plugin**. IGOR evaluates potential new fingers through its policy-plugins. Each new potential finger is evaluated with a value which is the result of a multi-

[1] http::/www.videogor.net

plication of all evaluation values of all participating policy plugins. Hence, not only one plugin decides if a new connection is opened.

Each of the policy plugins are in the position to evaluate a new potential connection with a value from 0 which means the worst to 1 which means the best. Also routing decision are evaluated.

- **Message-plugin**. Every plugin which uses own message types is a message-plugin. Here decisions of how to react on sending and receiving of certain message-types can be done. Hence, it is possible to react directly on message-types. Moreover, one can decide what happens with the received message.
- **Task-plugin**. The task-plugins are used to fulfill periodic actions. It is possible to select a certain time-value for the periodic execution.
- **Peek-plugin**. The peek-plugin is only used to consider the receiving or sending of messages. This is only possible here without direct impact on handling the message further.

6.2 Proximity Neighbor Selection

Within this section it is shown how the Merivaldi algorithm is realized as a policy-, message-, task-, and peek-plugin in more detail.

6.2.1 Message Types

Three new message types: GP-message, CND-message, and CNDA-message, are introduced.

- **GP-message (Gossip-message)**. This message type is responsible for the GOSSIP-protocol which was introduced in 5.2.1 on page 48. Within such a message new potential ring-members are published.
- **CND-message (Closest-Node-Discovery-message)**. This message type contains the Closest-Node-Job for a requester. It is forwarded by the receivers in the case if a closer node is found, as explained in 5 on page 43. It contains the requestor's id and a tracking of visited ids. This tracking is necessary because first tests showed that CND-messages can go astray into loops. The reason for this behavior is explained in 1 on page 52.
- **CNDA-messsage (Closest-Node-Discovery-Answer-message)**. This message contains the id-transport-address pair of a potential new finger.

6.2.2 Coordinates

Vivaldi is implemented in a way that different counts of dimensions are easy feasible. A class Coordinate contains all functions which are used in the Vivaldi specification [5]. The implemented Vivaldi-algorithm uses IGOR's PING-message type to communicate the coordinate and error estimate to the counterpart. To obtain the attendant RTT measurement the Linux kernel is queried: The Linux kernel maintains internal data structures that keep track of active TCP-connections and their parameters [34]. Hence, the needed information belonging to each other can be obtained in this way: The PING-message piggy-backs the coordinate and error estimate of the counterpart, and the attendant TCP-connection can be used for the RTT determination.

Each coordinate starts with zero per dimension and after a first RTT-measurement is done, it is pushed away a distance equal to the measured RTT from the PING-messages sender in a randomly chosen direction. The Mersenne Twister by Makoto Matsumoto et. al [35] and its implementation of Jason R. Blevins[2] is used to compute the random direction. The Mersenne Twister is used because it is one of the best random number generators.

Merivaldi's Vivaldi uses a smoothing component of measured RTTs. It is necessary to overcome the problem of outliners to avoid oscillation in the coordinates. In Merivaldi a RTT measurement of a TCP-connection is determined as the average of the values between the 25%-Percentile and the 75%-Percentile of the last 20 RTT measurements.

In the beginning, the average of the first 5 measurements is used. After this threshold is reached the described smoothing takes place.

6.2.3 Ring-Members

Ring-members are represented through an own class. A ring-member contains a node-reference[3], and a time-value which represents its relative online time. It should be mentioned that the former class noderef is extended by a Vivaldi coordinate and its error. Because of this extension each new reference suggestion can be evaluated by the basic proximity plugin. With it evaluation is possible whenever a reference suggestion is done by any plugin.

6.2.4 PNS and PRS

The Merivaldi-framework implementation consists of five main parts: RingManagment-Task, RingMemberUpdateTask, VivaldiPeek, ProximityMessaging and ProximitySelection-Policy.

[2] http://www.duke.edu/ jrb11/

[3] A node-reference consists of an identifier-transport-address, a vivaldi-coordinate and a vivaldi-error

1. **Part: RingManagementTask-Plugin**. Periodically ring-members are sent to random chosen ring-members in GOSSIP-messages. This was explained in 5.2.1 on page 48. Therefore a ring-member of each ring is selected per random. One of these is selected to be the receiver, the rest of these illustrate message content. Here the GOSSIP-messages are produced and sent periodically. Also, the GOSSIP-message handling is done here: If a GOSSIP-message is received, the potential new ring-members are included in the rings and the ring-management takes place as explained in 5.2.1 on page 47.

2. **Part: RingMemberUpdateTask-Plugin**. The online time-value of ring-members which are fingers, too, are updated periodically within this part. Also, the first ring-members are introduced by this plugin. Periodically the finger-ring-members are updated or in the case of not knowing them already, placed in the Merivaldi rings.

3. **Part: VivaldiPeek-Plugin**. The only "peeked" message is the PING-message. It is usually used for keeping TCP-connections alive. But in the Merivaldi implementation it contains additional data like the Vivaldi coordinate and the remote error-estimate. Whenever a PING-message is "peeked", the coordinates and the error are used in combination with the measurement of the RTT to adjust the own coordinate. A smoothing of the measured RTTs of one connection is done as explained in 6.2.2 on page 59 to avoid oscillation. But the drawback of every smoothing is that the reaction to new conditions needs some time.

4. **Part: ProximityMessaging-Plugin**. The Closest-Node-Jobs and attendant answers are produced, sent and handled here. A CND-message is produced and periodically sent to one ring-member per ring. The receiving ring-member is selected randomly. A Closest-Node-Job-Answer is produced when the node itself is already on the track of this Closest Node Discovery or if no better node can be found and if no appropriate connection already exists.

5. **Part: ProximitySelectionPolicy-Plugin**. Two different tasks are fulfilled by this plugin:

   - For Proximity Neighbor Selection it is tried to evaluate suggested new connections with the best latency characteristic out of the appropriate id-range as best. Therefore the appropriate TCP-connection entries are considered and their RTTs are compared to the suggestion's one. The number of different RTTs of these TCP-connections and the suggestion are counted. 1.0 is divided by the number of different RTTs. With it each reference finds its evaluation. So the one with the highest RTTs are evaluated with small values. The best one is evaluated with 1.0 which illustrates the best evaluation. The rest of the fingers is evaluated with a constant value. But here it is to say again: Not only this plugin is asked for evaluation. So it is also probable that the suggestion made from a CNDA-message is evaluated badly overall and no connection will be established.

- The second task takes Proximity Route Selection into account. The next hop is the result of an evaluation of all possible next hops which are a step closer in id-space. Hence, all possible route proposals are evaluated by their latency which can be directly determined by the opened TCP-connections as explained earlier in 6.2.2 on page 59. The Proximity Route Selection evaluation process is similar to the evaluation of a new potential connection.

Chapter 7

Testing Merivaldi

Within this section it is described how Merivaldi is tested for its functionality and effectiveness.

7.1 Measurement Details

7.1.1 Vivaldi

The integrated Vivaldi [5] algorithm is tested for its relative error development. Whenever a new PING-message is received the coordinates are renewed as described earlier in 5.2.3 on page 49. At the same time the determined RTT, the measured RTT, the absolute and the relative prediction error are written to a file.

More testing details. Within the following tests, the implemented Vivaldi uses 7 dimensions. A coordinate is updated whenever a PING-message is received. Each coordinate starts with (0,0,0,0,0,0,0). After a first measurement it is pushed away a distance equal to the measured RTT from the PING-message sender in a randomly chosen direction. Then the Vivaldi update functionality is processed whenever a PING-message is received.

7.1.2 CND-Hops

It is interesting how many CND-hops a CND-job needs until a closest node is found. Each CND-message contains a counter-variable which is incremented per CND-hop. A CND-hop should not be mistaken with a normal routing hop, i.e., a CND-hop can contain multiple routing hops. As described in 5.2.2 on page 48 the found closest node answers the requesting node only if no appropriate TCP-connection already exists. The CNDA-message, contains

the summed up CND-hop count of this CND-job. Whenever a CNDA-message is received the CND-hop count is written to a file.

Further implementation details. An additional variable is added to a CND-message. It is incremented whenever the CND-job is forwarded to its next destination.

7.1.3 Lookup Latency

A query for a random generated id is repeated after a certain time interval. A query message contains a variable which contains the summed up RTTs of the TCP-connections the query used, until it reached its final destination. Additionally, at each intermediate hop the hop-count is incremented . When a query is received by its final destination the summed up RTTs and the average latency of a query is written to a file.

Further implementation details. An additional task-plugin, "LookupTimeTask-plugin", is implemented which periodically queries for random ids. Therefore an additional LOOKUP-message type is implemented. The message's target is the mentioned random identifier.

7.1.4 Routing Table Latency

As described in 6 on page 57 the routing-table consists of opened TCP-connections. The RTTs of these TCP-connections are considered here. A fixed time period the average, the 25%-quartile, the median and the 75%-quartile is determined and written to a file.

Implementation details. An additional task-plugin, "ImprovementConsiderationTask-plugin", is implemented. Here the the average, the 25%-quartile, the median and the 75%-quartile is determined and written to a file as mentioned above.

7.2 PlanetLab

Testing a distributed system with real world conditions is difficult if one can not use lots of distributed hosts in the world. This is the reason why PlanetLab has been founded in 2003. PlanetLab is a research network that currently consists of 784 nodes at 82 sites distributed over the globe [6]. PlanetLab has been used by lots of researchers at academic institution or industry to test and develop distributed systems like distributed storage, network mapPING, Peer-to-Peer systems and so on [6].

7.3 Tested IGOR Versions

Different kinds of IGOR are tested with different aims.

1. Version: IGOR with Merivaldi and with PRS.
   (a) Number of hops a CND-message needs until a CNDA-message is sent.
   (b) The development of latency in the routing table.
   (c) The development of lookup latency.
2. Version: IGOR with Merivaldi and without PRS.
   (a) Number of hops a CND-message needs until a CNDA-message is sent.
   (b) The development of latency in the routing table.
   (c) The development of lookup latency.
3. Version: IGOR without Merivaldi and with PRS.
   (a) The development of latency in the routing table.
   (b) The development of lookup latency.
4. Version: IGOR without Merivaldi and without PRS.
   (a) The development of latency in the routing table.
   (b) The development of lookup latency.

7.4 Testing Environment

IGOR is tested on PlanetLab with 83 hosts. Each described version starts 3 IGOR instances with a single bootstrap IGOR-instance at the same time. To exclude different network conditions it is tried to start the IGOR-instances the same time. The testing period lies between 3 and 4 hours.

Emerging Problem: Using a finger-number restriction, a lot of IGOR-instances stop and no stable IGOR overlay network is established[1]. The number of existing IGOR-nodes converges to the number of allowed fingers at maximum. Sadly this error could not be fixed in time to offer good results.
Solution: It is used a finger-restriction of 50 allowed fingers at maximum. This is the normal configuration for an IGOR-instance. Despite this accommodation a lot of IGOR-instances stopped working as described. This is considered until each IGOR-instance provides overall knowledge about all the others. Finally this is an optimal system and no improvement is

[1] A stable IGOR overlay consists of performing IGOR-instances

possible. Thereby not every test result is considered. This error is described more precisely in appendix.

7.5 Results

Within this section the results are illustrated and analyzed.

7.5.1 Vivaldi

The development of the latency-prediction for a single arbitrary node which was alive until the test stopped, is shown in Figure 7.1 on page 65. An Accumulation of all IGOR-instances in Figure 7.2 on page 66. Here chronological succeeding 100 measurements are tied together and the median of the relative prediction error is determined. Hence, each point illustrates the median of 100 succeeding measurements in time. Within the Figure which accumulates all IGOR-instances, the medians of the same point are summed up and averaged.

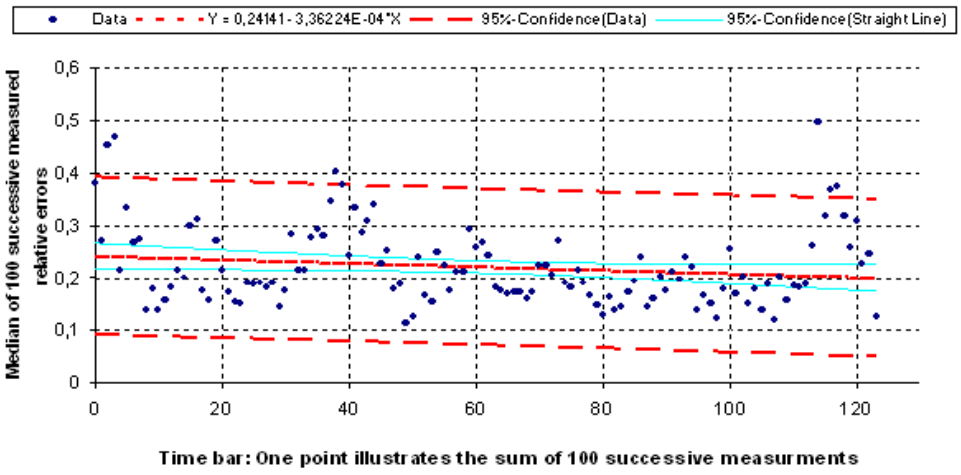

Figure 7.1: The Figure shows the development of latency-prediction. Each 100 successive measurements the median of the relative error of these is determined and drawed. The improvement of latency-prediction is directly illustrated with the regression-line.

One can easily recognize that the prediction of latency in IGOR develops quite well as the regression-line shows. It is confirmed that Vivaldi works with high quality with the use of the PING-message type and 7-dimensions within a euclidean coordinates space. Despite the described IGOR-error in 7.4 on page 64 this observation can be kept in such a case.

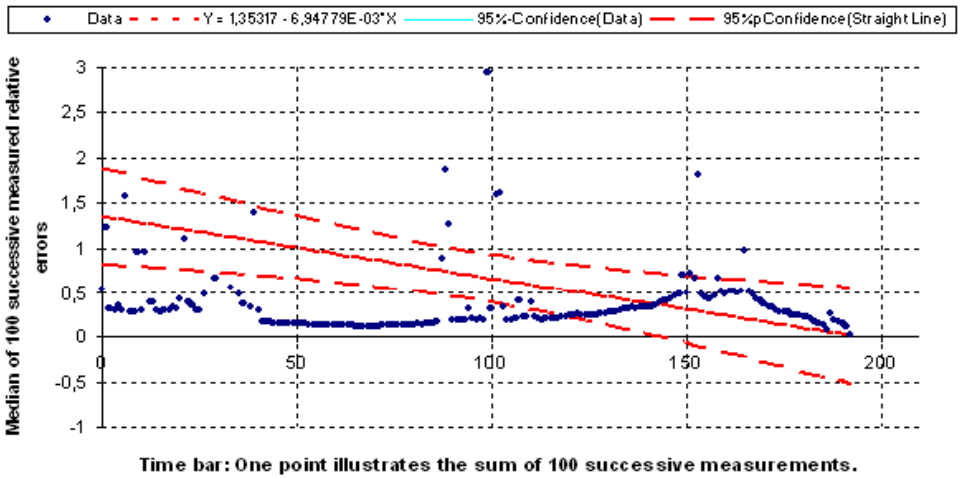

Figure 7.2: The Figure shows the development of latency-prediction. Each 100 successive measurements the median of the relative error of these is determined and plotted. It is easy to recognize the improvement of latency-prediction which is directly illustrated with the regression-line.

7.5.2 CND-Hops

All IGOR-instances which implement Merivaldi are considered here. Not more than 12 CND-hops are done until a closest node is found. This is shown by the Figure 7.3 on page 67 The little more than 8 hops which one expect can be a result of changing network conditions, not good adjusted coordinates and an implementation characteristic: In Merivaldi a node considers all ring-members of each ring when querying for a closer one. Additional it is to say the earlier mentioned IGOR-error falsifies this histogram in its distribution.

7.5.3 Lookup Latency

It is not easy to compare two different lookup-delays if the lookup is done for different destinations, one does not know before. Earlier tests showed nodes leave the overlay and a stable network cannot be established. More in 7.4 on page 64. Additionally a real testing environment to determine if a lookup is successful is missing up to now. Hence, the responsibility areas are displaced. Lookups which used , e.g., 2 hops firstly, need only one hop.

Therefore to extract really good comparisons when testing PRS it is necessary to have a testing environment which clamps if a responsibility area has changed. In addition the size of the network should stay equal. Therefore a lookup-consideration is more-or-less

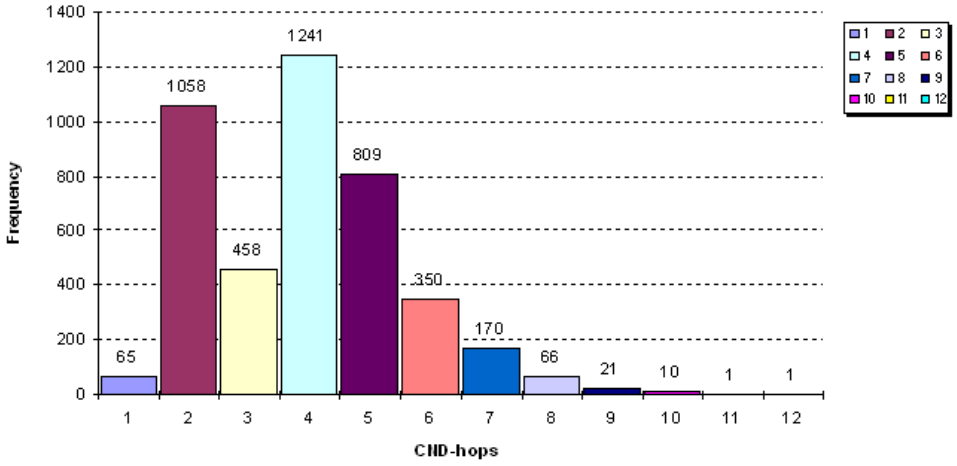

Figure 7.3: The Figure shows the frequency of different numbers of CND-hops of all IGOR instances using Merivaldi. One can observe 12 CND-hops are performed at maximum.

senseless and is not traced further.

7.5.4 Routing Table Latency

The development of the RTTs in the routing table is analyzed here in more detail. The average, the 25%-Percentile, the Median, and the 75%-Percentile of the summed up RTTs should decrease when fingers are exchanged with latency-closer nodes. If fingers are not exchanged the gradients of these values should be lower than in a system using no PNS.

Unfortunately no improvement can be found. The reason for this behavior: Both IGOR test networks are converging to an optimum where no improvement is possible. This is due to the error described in 7.4 on page 64. Additonally this Figure shows a better latency development for the IGOR version without Merivaldi. This behavior can be a result of Churn in the "IGOR with Merivaldi" overlay network. Especially IGOR-instances, which represented good latency characteristics in this IGOR-version can be lost. In the other version they are not. One can imagine two IGOR-overlays on different hosts. One is distributed on nearby hosts and the other one not. Comparing the two leads to falsified pictures.

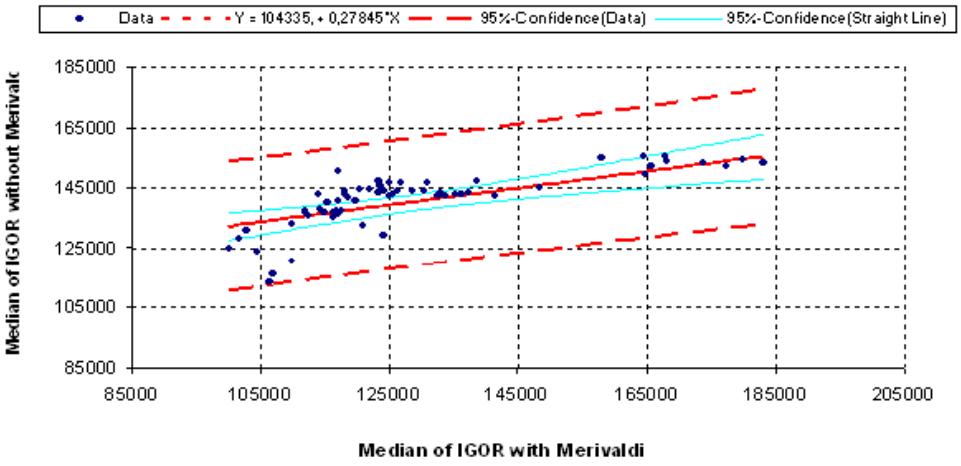

Figure 7.4: This Figure shows a direct comparison between the median RTT of the routing-tables of all IGOR-instances of two different versions: IGOR with Merivaldi and IGOR without Merivaldi. The median of 10 successive measurements is summed up for each point. This is done for all nodes. Finally the average of the medians of the same measurement points is determined.

Chapter 8

Conclusion

In this diploma thesis the IGOR [1], [2], [3] overlay network was examined to implement Proximity Neighbor Selection and Route Selection strategies. A new approach of Proximity Neighbor Selection was developed and introduced: Merivaldi illustrates a combination of the Meridian[4] framework and the Vivaldi[5] algorithm for estimating Internet latency between arbitrary Internet hosts.

Merivaldi is quite similar to Meridian. It differs in that it uses no direct Round Trip Time measurements like Meridian does to obtain latency characteristics between hosts. Merivaldi obtains latency characteristics of nodes using the latency prediction derived from the Vivaldi-coordinates. A Merivaldi-node forms exponentially growing latency-rings. The rings accord to latency distances to the Merivaldi-node itself. In these rings node-references[1] are inserted regarding to their latency characteristics. These node-references are obtained through a special protocol. A Merivaldi-node finds latency-closest nodes through periodic querying its ring-members for closer nodes. If a closer node is found by a ring-member the query is forwarded to this one until no closer one can be found. The closest node on this way reports itself to the Merivaldi-node.

Merivaldi was established to be only a modest burden for the network and system. Merivaldi was implemented in IGOR and partially tested on PlanetLab. The average hop-count a Merivaldi CND-job needs lies within the borders of $O(log\ N)$ where N is the number of participating nodes. This result of analysis work is confirmed by empirical tests. The test used 252 IGOR-instances with 9 rings and 7 ring-members each. The IGOR-instances have been placed on different PlanetLab nodes. A maximum of 12 CND-hops is not overstepped within this test.

The few extra CND-hops which one might detect can be a result of changing network conditions and an implementation characteristic: IGOR's Merivaldi considers all nodes of every ring when querying for a closer one. Merivaldi's Vivaldi implementation uses

[1] Here, a node-reference means not a direct node but only a "pointer" to a node.

IGOR's PING-messages, usually used for keeping TCP-connections alive and direct TCP-measurements from the Linux kernel. With it Merivaldi's Vivaldi offers good prediction results. To avoid oscillation caused by RTT outliners Vivaldi also uses a special developed smoothing.

Unfortunately it was not possible to achieve a stable IGOR network. Because of an error the established IGOR-overlay converged to an optimal overlay where each node knows each other. Future research work on Merivaldi should consist of testing Merivaldi with the use of simulators to obtain high quality results of its effectiveness. Whether Merivaldi finds all closest nodes is a further crucial question. Additionally a cohesion of Vivaldi's prediction error with the developing latency distribution in the routing table of a Merivaldi node can be a further field of research interest.

Appendix A

Appendix

A.1 Configuration of Merivaldi in IGOR

Every IGOR-instance uses a file igor.conf to use a certain parametrization. IGOR's Merivaldi is configurable through this file, too. For example, the igor.conf file looks like this:

```
timer_stabilize: 15
timer_fixfingers: 20
timer_pinger: 30
cont_if_no_conns: 1
max_conns_allowed: 50
min_conns_wanted: 10
vivaldi_dimensions: 7
number_of_rings: 9
number_of_ringmembers: 7
timer_ring_management: 20
timer_ring_member_update: 20
timer_lookup: 10
timer_improvement_consideration: 20
merivaldi: 1
prs: 1
pns_test: 1
debug: DFLT:5,CBL:5,NODE:5,CONN:5,PARSE:5,MESG:5,
CFG:5,ID:5,DBG:5,RAWMSG:5,BUF:5
```

IGOR's Merivaldi uses different parameters from the igor.conf file:

- If Merivaldi should be used:"merivaldi: 1". If Merivaldi should not be used: "merivaldi: 0".

- If PRS should be used: “prs: 1”. If PRS should not be used: “prs: 0”.
- The number of Vivaldi dimensions which should be used: vivaldi_dimensions.
- The number of rings which should be used: number_of_rings.
- The number of ring-members per ring which should be used: number_ring_members.
- The timer of when the ring-management should take place: timer_ring_management.
- The timer of when the ring-members-update should take place: timer_ring_member_update.

For testing reasons the igor.conf file contains some more configuration parameters:

- If a test is desired with the test-plugins, LookupTaskPlugin and ImprovementConsideration: “pns_test: 1”. If not, “pns_test: 0”.
- The timer when a lookup is performed: timer_lookup.
- The timer the routing table is considered: timer_improvement_consideration.

A.2 Error

The emerged error, described in 7.4 on page 64 which was considered during the tests, is described here: Every IGOR-instance is able to produces a logfile. For instance, the last rows of the logfile of an IGOR-instance which stopped working are presented. The IGOR-process stops without any node-shutdown[1] procedure as one can see in the last part of the logfile:

```
1180851309.641382 26862 CONN4:read(fd -1): Bad file descriptor
1180851309.641435 26862 DFLT5:cRoutingTable::BrokenConnection(cone=0x813cde8): removing connection
1180851309.641452 26862 CBL5:can't unregister callback with bogus handle 135208624
```

If one traces the source code one can conclude the IGOR-instance tries to read from a dead connection. Therefore one can find the “Bad file descriptor”. In addition this connection should be unregistered to be not callbacked again. Maybe it is tried to unregister a connection which do not exists anymore. Sadly, this error could not be solved until now.

This leads to the convergence of the number of IGOR-instances to the max_conns_allowed configuration parameter. In this case any IGOR-instance has overall knowledge about all other existing IGOR-instances. Among other tests, 32 IGOR-instances on just one host with max_conns_allowed set to 15 have been started. After a while only 15 IGOR-instances are still working. It is different when IGOR-instances get lost. Sometimes this happens

[1] An IGOR-instance usually stops working with definite node-shutdown. This is usually written to the logfile.

fast. Sometimes this needs sometime. Because of this error, some desired tests could not be performed with quality results.

List of Figures

Bibliography

[1] Kendy Kutzner and Thomas Fuhrmann. The IGOR File System for Efficient Data Distribution in the GRID. In *Proceedings of the Cracow Grid Workshop CGW 2006*, Cracow, Poland, 2006.
http://i30www.ira.uka.de/research/documents/p2p/2006/kutzner06igor.pdf.

[2] Ivan Kostov. Study-Thesis: Developing an Automated Test Environment for the Overlay Network IGOR, 2006.

[3] System Architecture Group. The System Architecture Group at the University of Karlsruhe, 2006.
http://i30www.ira.uka.de/p2p/.

[4] B. Wong, A. Slivkins, and E. Sirer. Meridian: A Lightweight Network Location Service without Virtual Coordinates, 2005.
http://www.sigcomm.org/sigcomm2005/paper-WonSli.pdf.

[5] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A Decentralized Network Coordinate System, 2004.
http://www.sigcomm.org/sigcomm2004/papers/p426-dabek111111.pdf.

[6] Planetlab, 2007.
http://www.planet-lab.org.

[7] Frank Dabek. *A Distributed Hash Table.* PhD thesis, Massachusetts Institute of Technology, November 2005.
http://pdos.csail.mit.edu/papers/fdabek-phd-thesis.pdf.

[8] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. pages 149–160.
http://pdos.csail.mit.edu/papers/ton:chord/paper-ton.pdf.

[9] Simon Rieche, Heiko Niedermayer, Stefan Goetz, and Klaus Wehrle. *Peer-to-Peer Systems and Applications*, volume 3485 of *Lecture Notes in Computer Science, LNCS*, pages 79–93. Springer, Heidelberg, Germany, September 2005.

[10] A. Oram. *Peer-to-Peer - Harnessing the Power of disruptive Technologies*. OŔeilly, 2001.

[11] Gnutella, 2006.
http://www.gnutella.com.

[12] David Karger, Eric Lehman, Tom Leighton, Mathhew Levine, Daniel Lewin, and Rina Panigrahy. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In *ACM Symposium on Theory of Computing*, pages 654–663, May 1997.
http://theory.lcs.mit.edu/˜ karger/Papers/web.ps.gz.

[13] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A Scalable Content Addressable Network, 2000.
http://www.sigcomm.org/sigcomm2001/p13-ratnasamy.pdf.

[14] Antony Rowstron and Peter Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, 2001.
http://research.microsoft.com/˜ antr/PAST/pastry.pdf.

[15] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An Infrastructure for fault-tolerant wide-area Location and Routing. Technical Report UCB/CSD-01-1141, Computer Science Division, U. C. Berkeley, April 2001.
http://www.cs.cornell.edu/people/egs/cs615-spring07/tapestry.pdf.

[16] Dahlia Malkhi, Moni Naor, and David Ratajczak. Viceroy: A Scalable and Dynamic Emulation of the Butterfly. In *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing*, 2002.
http://www.wisdom.weizmann.ac.il/˜ naor/PAPERS/viceroy.pdf.

[17] Kendy Kutzner, Curt Cramer, and Thomas Fuhrmann. A Self-Organizing Job Scheduling Algorithm for a Distributed VDR. In *Workshop: Peer-to-Peer-Systeme und -Anwendungen, 14. Fachtagung Kommunikation in Verteilten Systemen (KiVS 2005)*, Kaiserslautern, Germany, 2005.
http://i30www.ira.uka.de/research/publications/p2p/2005/kutzner05dvdr.pdf.

[18] D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, B. Zhao, and J. Kubiatowicz. Oceanstore: An extremely wide-area Storage System, 2000.
http://oceanstore.cs.berkeley.edu/publications/papers/pdf/oceanstore-tr-may99.pdf.

[19] Paul Francis, Sugih Jamin, Cheng Jin, Yixin Jin, Danny Raz, Yuval Shavitt, and Lixia Zhang. IDMaps: A Global Internet Host Distance Estimation Service, 2000.
http://idmaps.eecs.umich.edu/papers/ton01.pdf.

[20] E. Ng and H. Zhang. Predicting Internet Network Distance with coordiantes-based Approaches, 2001.
www.cs.umd.edu/class/spring2007/cmsc711/papers/gnp.pdf.

[21] K. Gummadi, S. Saroiu, and S. Gribble. King estimating Latency between Arbitrary Internet End Hosts. 2002.
http://www.mpi-sws.mpg.de/˜ gummadi/papers/king.pdf.

[22] R. Cox, F. Dabek, F. Kaashoek, J. Li, and R. Morris. Practical Distributed Network Coordinates, 2003.
http://pdos.csail.mit.edu/˜ rtm/papers/hotnets-vivaldi.pdf.

[23] Frank Dabek Robert Cox. Learning Euclidean Coordinates for Internet Hosts, 2002.
http://pdos.lcs.mit.edu/˜ rsc/6867.pdf.

[24] A. Nandan, M. Parker, G. Pau, and A. Nader-Teherani. Intelligent Neighbor Selection in P2P with CapProbe and Vivaldi. In *Proceedings of IEEE Globecom, St. Louis, IL*, 2005.
http://www.cs.ucla.edu/˜ gpau/docs/Gpau-Application-Pkg.pdf.

[25] Peter Pietzuch, Jonathan Ledlie, Michael Mitzenmacher, and Margo Seltzer. Network-Aware Overlays with Network Coordinates. In *ICDCSW '06: Proceedings of the 26th IEEE International ConferenceWorkshops on Distributed Computing Systems*, page 12, Washington, DC, USA, 2006. IEEE Computer Society.
http://dx.doi.org/10.1109/ICDCSW.2006.76.

[26] Sylvia Ratnasamy, Scott Shenker, and Ion Stoica. Routing Algorithms for DHTs: Some Open Questions. 2002.
http://www.cs.rice.edu/Conferences/IPTPS02/174.pdf.

[27] M. Castro, P. Druschel, Y. Hu, and A. Rowstron. Exploiting Network Proximity in Distributed Hash Tables, 2002.
http://www.cs.rice.edu/˜ druschel/publications/fudico.pdf.

[28] R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The Impact of DHT Routing Geometry on Resilience and Proximity, 2003.
http://www.sigcomm.org/sigcomm2003/papers/p381-gummadi.pdf.

[29] F. Dabek, J. Li, E. Sit, J. Robertson, M. Kaashoek, and R. Morris. Designing a DHT for low Latency and high Throughput, 2004.
http://project-iris.net/irisbib/papers/dhash:nsdi/paper.pdf.

[30] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling Churn in a DHT, 2004.
http://srhea.net/papers/bamboo-usenix.pdf.

[31] Hancong Duan, Xianliang Lu, Hui Tang, Xu Zhou, and Zhijun Zhao. Proximity Neighbor Selection in Structured P2P Network, 2006.
http://doi.ieeecomputersociety.org/10.1109/CIT.2006.154.

[32] Yingwu Zhu and Xiaoyu Yang. Implications of Neighbor Selection on DHT Overlays. *mascots*, 0:197–206, 2006. http://doi.ieeecomputersociety.org/10.1109/MASCOTS.2006.27.

[33] L. Tang and M. Crovella. Virtual Landmarks for the Internet, 2003. http://www.imconf.net/imc-2003/papers/p320-tang1.pdf.

[34] Rene Pfeiffer. Measuring TCP Congestion Windows. 2007. http://linuxgazette.net/136/pfeiffer.html.

[35] Makoto Matsumoto and Takuji Nishimura. Mersenne Twister: A 623-dimensionally equidistributed uniform pseudo-random Number Generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, 1998. http://doi.acm.org/10.1145/272991.272995.