INSTITUT FÜR INFORMATIK
Technik sozialer Netzwerke

Universitätsstr. 1 D–40225 Düsseldorf

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Design, Implementation and Evaluation of a Motion-aware location-based Peer-to-Peer Overlay

Ivaylo Radev

Masterarbeit

Beginn der Arbeit:	25. August 2015
Abgabe der Arbeit:	25. Februar 2016
Gutachter:	Jun.-Prof. Dr. Kalman Graffi
	Prof. Dr. Martin Mauve

Erklärung

Hiermit versichere ich, dass ich diese Masterarbeit selbstständig verfasst habe. Ich habe dazu keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Düsseldorf, den 25. Februar 2016

_____

Ivaylo Radev

Abstract

The main role of the human in the extinction of a lot of animals defines the need of good telemetry solution in order to preserve and protect the wildlife. The distribution of the animals around the world makes the use of client-server architectures almost impossible for telemetry purposes. In this context peer-to-peer overlays provide a good alternative to centralized structures. In this Thesis we discuss the different ideas and solutions used in location-based peer-to-peer networks. As none of the solutions consider movement of the peers, which is very important in the animal telemetry, we introduce GeoP2P.Extended, a novel location-based motion-aware overlay, which provides the movement algorithm and a structure for following the movement behavior of the participiating peers. As it is based on a tree structure, different methods for maintance of this tree structure are included. Our novel solution is then compared with the existing solutions, even if this is only possible in its static state, where peers are not moving. In this static state, but also if peers are moving, the GeoP2P.Extended stays stable and its performance is comparable with the other introduced solutions. Finally, the different configuration parameters of the overlay are evaluated in order to define their impact on the behavior of the overlay and to find out the best configurations for the different real life situations.

Acknowledgements

I am especially greatful to my wife, Miglena Mihaylova, and to my parents, Svetla and Lilyan Radevi, for the great support in all the hard times, that I had during and before writing this Thesis. Writing this Thesis would not be possible without their care and cover in this time.
I want to thank to the people in my life, that support me and believe in me every single moment - my sister, Zhaneta Radeva, my best friend Alexandra Daskalova and also my great parents, Zhivka, Tonka, Ivan and Gancho.
I would also like to thank to my great supervisor, Tobias Amft, for all the patience, support and all the tips, that he gave me.

Thank you!

Contents

List of Figures

Listings

1 Introduction

There are around 15000 species of mammals, 6000 species of reptiles, 9000 birds, 1000 amphibians , 20 000 species of fish and over 1 million recorded species of insects over the world. The most powerful of all is the human. The tragedy is that the human has also the most important role in the extinction of the other species. There are 764 species, that are extinct or extinct in the wild in the year 2015. According to the International Union of Conservation of Nature and its Red List , there are another 11982 species that are vulnerable, endangered or critically endangered by 2015 [10] .

1.1 Motivation

Organisations like IUCN (International Union of Conservation of Nature) or stand alone scientists and researchers have an important role in the preservation and protection of the wildlife. To have the chance to save the species, their information about the animals is needed. The most important and powerful tool for collecting this information is the telemetry. The telemetry is using different techniques to obtain data about the behavior of the animals as they move in their habitat. All of them use tags that are either archiving or transmitting the information to satellites or data stores. According to the US National Oceanic and Atmospheric Administration (Link) there are three different categories of telemetry, depending on the method of data recovery: archival, satellite and acoustic. The simplest solution is the archival. In this solution, all the information is archived in a tag and can be recovered later, when the scientists have physical access to the tag. This is most of the time not so easy and predictable, but it gives also the most detailed data. The next category, the satellite, is obtaining the data in real-time. The biggest problem here is the limited bandwidth and thus only a short summary of the data can be recovered. There is also big advantage - the tag does not need to be physically accessed. The third form of telemetry is the acoustic. Acoustic telemetry is used for studying species that are too small to carry the tags. It is divided in active and passive acoustic telemetry.
All three categories of telemetry have advantages and disadvantages. One of the earlier disadvantages of the archival and satellite telemetry, the size of the tags and their placement on the animals, is nowadays solved. As the tag can not be heavier than 5 percent of the weight of the animal, telemetry in the past was only usable on big mammals like dolphins or elephants. In the last years a lot of companies started to produce small tags for animals like rats or birds [11]. However these solutions are increasing the other problems of the telemetry. As already mentioned the archival telemetry is providing a lot of data to the scientists, but has the disadvantage that the information can be obtained only at tags retrieval. The satellite telemetry at the other side can provide information in real-time, but it is not applicable if there is no direct visual contact to the satellite and trough the limited bandwidth it can not provide all the information needed. The new small tags solve none of this problems. They still need visual contact to the satellite or physical contact to the scientist to deliver the data. As they have to be small they can not provide a lot of memory to save all the information and the only data saved on them is an animal ID, battery life and temporary information about the position of the animal.

Some researchers have proposed the idea to build heterogeneous wireless networks to mix the different categories of telemetry in one network. One of them is EcoLocate [1]. EcoLocate is working like shown on Figure 1.

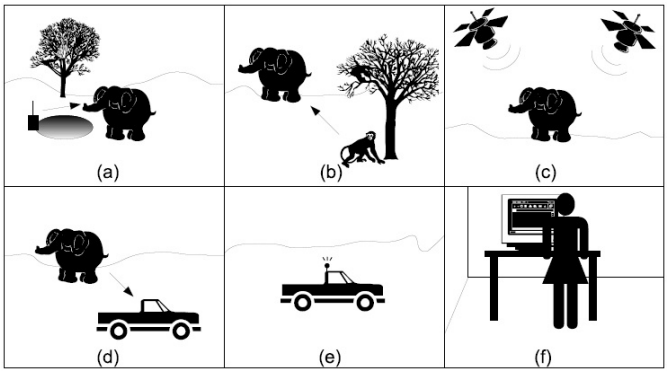


Figure 1: EcoLocate - tracking and telemetry system. Source: [1]

First a stationary sensor is sending the data to an animal that is near it. After that on 1b) the animal, in our case the elephant, is walking trough its area and passes by another animal - the monkey. The monkey tag is contacting the tag of the elephant and uploads data about the monkey over the last days. At 1c) the elephant tag is connecting to a GPS satellite and is determining its location. Thus the approximative position of the monkey can be retrieved. A vehicle driving trough the elephants area retrieves the data from the elephant tag at step d) and saves it to its memory. The data is transmitted to the park ranger in the last 2 steps. Thus EcoLocate combine the advantages of the archival and satellite telemetry. One problem can still not be solved by EcoLocate. The elephant or another big animal, with big tag, is still needed to collect the information from the small animals and send it to the rangers. If the elephant tag get lost or just fails, all the information about the monkey and all other small animals saved on the elephant tag will be lost. Thus saving big amount of information at one or more big animals tags has to be avoided.

Comparing to other areas of use of networking, the best solution for this problem are the peer-to-peer networks. In peer-to-peer networks there is no need of centralized servers, as the tasks and the data is distributed between the participants. In order to use peer-to-peer networks for the situation with the animals and their tags, a location-based peer-to-peer networks will be needed. Furthermore we need to consider the movement of the participants, in our case animals and scientists, over the area covered by our network.

In this Thesis we will analyze existing location-based peer-to-peer overlays and the main techniques for indexing the spatial data, used in them, comparing their advantages and disadvantages.

1.2 Application area

As already mentioned, the main application area of our overlay will be finding objects, identified by their geographical coordinates. In our case these objects are the animals and also the researchers, following them. However there is a lot of other areas of the real life, where such an overlay can be used. It can be used by taxi drivers to locate the nearest customer or also for emergency services, where the casualty can inform e.g. the fire department about an accident and all the police and fire brigade teams can determinate which one is the nearest without using a centralized dispatcher. This will result in preventing deny of service because of overload in case of critical situation, such as the earthquake in Fukushima or the terrorist attacks in Paris.

The main point of this Thesis is to consider also scenarios with mobile users, as all the already mentioned application cases have mobile devices as part of it. Both animals and researchers in the first case are moving almost all the time and also the taxi driver and the police cars should be considered as mobile devices.

1.3 Requirements

When we take into account the application areas, that we have mentioned in Section 1.2 some main requirements become apparent.

Peer in area - First we have to have the possibility to find a peer in selected geographical area. In our example with the animals and the researchers, in most of the cases the researchers do not know the exact position of the animals. Thus they can only find them, if they start a global search for the ID of the animal in the area, that they expect the animal to be in.

All peers inside an area - The second requirement is to find all the peers in a selected area. This is needed for the case, that a scientist wants to collect the information from the sensors of all the animals in a specific area, e.g. national park. Furthermore the query can be more specified, allowing the scientist to find only all animals of one type in the selected area, e.g. all the lions in the park.

According to Kovacevic in [8] there are also some other requirements, caused by the nature of the peer-to-peer networks. In peer-to-peer networks the peers are leaving and joining most of the time. In our case they are also moving from one geographical point to another. This is a big challenge for the management of the load balance and also the stability of the network. It can also cause large amount of sent messages and also changes in the size of the network.

Completeness and correctness - Considering this, we can set the completeness and correctness of the retrieved results as next requirement. This one is very important as complete and correct information is a precondition for the successful work in preserving the wild animals. The researchers can not afford to miss animals in the selected area. However as there is no central view of all the peers in the network, this requirement is pre-

senting a big challenge for the designer of the peer-to-peer networks.

Performance and management of the costs - management of the costs and performance is another important requirement. As both depend from each other, it is not possible to improve the performance without increasing the costs, whether for every peer or for the whole overlay.

Stability - A peer-to-peer overlay has to be also stable, meaning that the system stays persistent in case of changed conditions. Good example for changing conditions is joining and moving of peers or a large number of queries in a short time window.

Scalability - Last, but not least, as the number of peers in the network will change all the time, it is very important,that the overlay has to have the ability to adapt to this changing number of peers.

Considering all the above information the main requirements for a location-based peer-to-peer overlay are finding a peer in a selected area, search for all peers in a location, complete and correct results of the queries, good stability and scalability. Further requirements for location-based peer-to-peer overlay, are discussed and described in [8] and [12].

1.4 Outline

In this Chapter the need of motion-aware location-based peer-to-peer overlays was motivated. Further the requirements for such overlays were introduced. The next Chapters of this Thesis are structured as follows:

In Chapter 2 we present the different techniques for indexing of spatial data in Section 2.1. In Section 2.2 and Section 2.3 some of the current approaches, used by the scientists in the research of location-based peer-to-peer networks, are presented. At the end of this Chapter, two solution, that are very similar to our work, are presented in Section 2.4 and Section 2.5.

Chapter 3 presents our novel motion-aware location-based overlay, named GeoP2P.Extended. In Section 3.2 the design of the novel overlay is introduced. As our overlay is considering movement of the peers, a movement algorithm is introduced in Section 3.4.2.

In Chapter 4 we present a short overview of the simulator, that has been used for the implementation and evaluation of our overlay. Furthermore the design of GeoP2P.Extended in Section 4.2. After that the actual implementation of the overlay is described in Section 4.2.

In Chapter 5 we evaluate our overlay and compare it with the already presented location-based overlays. As they do not consider movement of the peers, separate evaluation of the novel overlay in case of motion of the peers is done. The metrics used for the evaluation are briefly overviewed in Section 5.1. Further in Section 5.2 the different simulation

scenarios are introduced. At the end, in Section 5.3, the results of the simulations are presented.

Chapter 6 gives an overview of this Thesis and introduces the future work with our motion-aware location-based overlay, which is not described in the Thesis.

2 Related work

2.1 Spatial data indexing

Most of the location based peer-to-peer overlays use one of the following techniques for indexing the spatial data , that we will introduce here.

2.1.1 Space partitioning

Space partitioning has as base an old idea of the early 70's. The scientist found out, that if a large amount of symbolic data is sorted alphabetically and stored in an array, then if using a binary search algorithm we can find in $\log_2 n$ operations if a new element is already in the list. First it was expected, that $n/2$ operations are needed if using sequential search. But there was also a need of dynamic structure, for example for adding and removing elements while working with the list of elements. Probably the best solution for this problem, especially in the peer-to-peer overlays are the binary space partition trees, the KD-trees and the R-trees. [13]

The binary space partition tree is a data structure, that recursively divides object or collection of objects by hyperplanes. This has to be done until some requirements are fulfilled. As already mentioned the idea is very old and was also first used in the 70's in the computer graphics. Later it was the base for the modern computer games like Quake where it helped to significantly increase the game's performance.

On Figure 2 we can see this technique over a random plane. Node *a* in the tree is representing the whole plane. In the second step the entire plane is divided into subplanes *b* and *e*, which we can see in the second level of the binary space partition tree. At level 3 of the tree the division of subplane *b* in *c* and an empty subplane and of subplane *e* in subplane *f* and an empty subplane is represented. This is going further until the conditions are fulfilled.

The KD-tree is a subclass of the binary space partition tree, but with two main differences. First all the planes in the KD-tree has to be perpendicular to the coordinate axis. Second and probably more important - the nodes in the tree are not representing any subplane of the main plane. They are just representation of points in the subplanes, such that each point is part of its own space.

Figure 3 shows us an example of KD-Tree. First the point A is causing a split, that is parallel to the x axis. A is then also assigned as a root of the tree. At the second level of the tree we have the nodes B and C, which are lying on the two correspondent lines that parallel to the y axis. At the third level of the tree there is only one node - D, representing the splitting line that is parallel to the x axis. The last two splitting lines are parallel to the y axis and have points E and F.

The R-Tree is a different structure. The "R" in the R-tree means rectangle. This abbrevia-

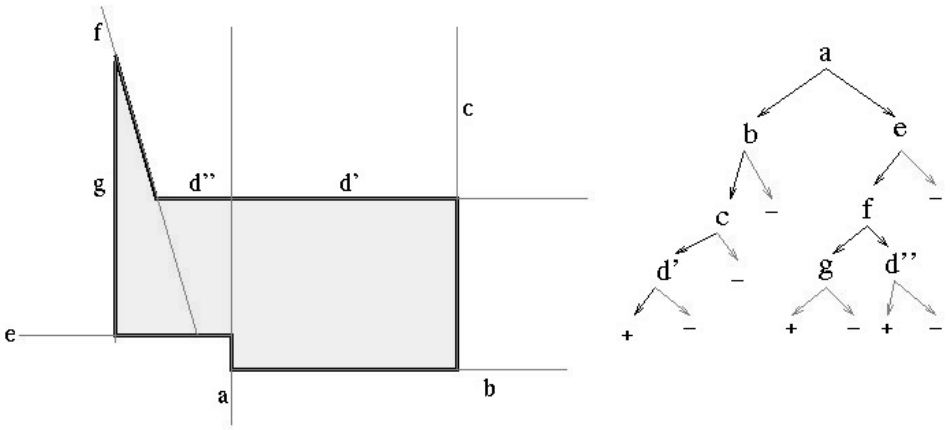


Figure 2: Binary space partition tree [2]

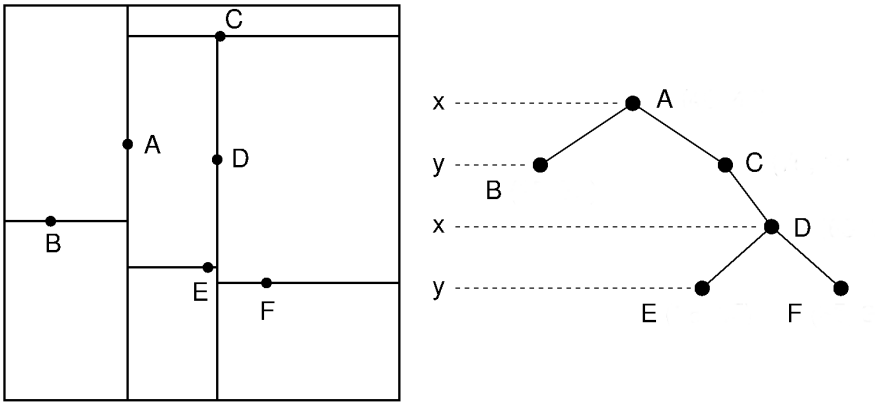


Figure 3: KD-tree [3]

tion is explaining the main idea behind the R-tree. Unlike the binary space partition tree it is not dividing the whole plane into subplanes. The R-tree is balanced search tree, that is organizing nearby objects in groups and forming minimum bounding rectangles(MBRs), which may overlap. The example of a R-tree in Figure 4 is showing the MBRs and their relations.

The main idea is to use the boundaries of the MBR to find out if a query intersect any of the objects in the MBR. This can be true, only if the query intersect the boundaries. R-trees are performing very well with real world data.

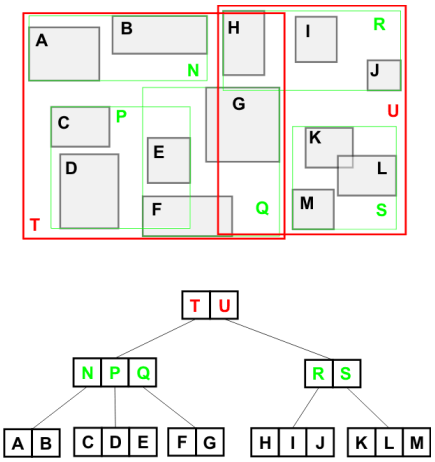


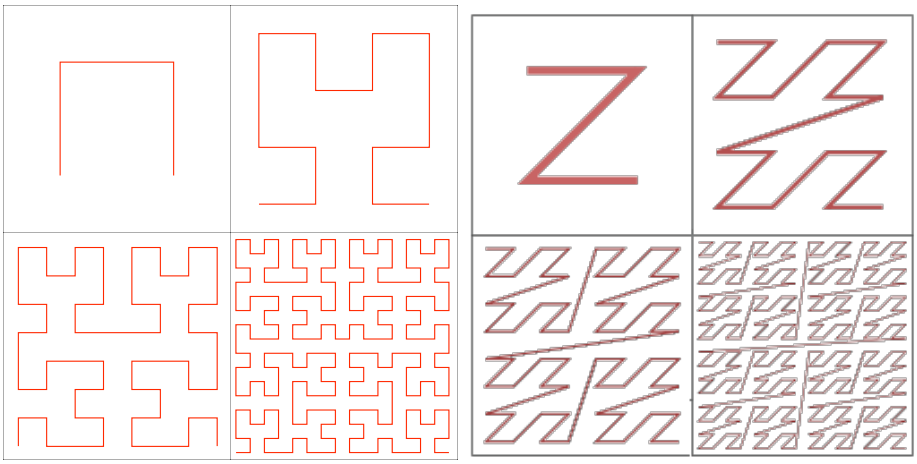
Figure 4: R-tree [4]

2.1.2 Space filling curves

The other technique for indexing spatial data are the space feeling curves. Space filling curve is a continuous surjective function that maps 1-dimensional space onto a higher-dimensional space, such as the units square, cube, hypercube, etc. In most of the cases the space filling curve divides the higher-dimensional space into smaller tiles of similar shape, together with a definition of the order in which these tiles should be traversed by the curve. [14]

In the indexing of spatial data we need to find such a space filling curve, that the distance between two point on the curve reproduces the between them in the higher-dimensional space. Unfortunately, " it is impossible to create a space filling curve, such that all points close in multi-dimensional space, are close on the curve, too. " [15]

We can see the Hilbert and z- space filling curve, respectively in Figure 5a and Figure 5b. Both space filling curves cover more points in the space with increasing number of iterations. Both curves have their advantages and disadvantages. The Hilbert space filling curve is more complex, but is covering better the space and because of this is very often used for high-dimensional spaces. The z space filling curve at the other side is very simple to understand and easy to use. That is the reason, why it is often preferred. Lawder and King analyze the space filling curves and their use for multidimensional indexing in [16] .



(a) (b)

Figure 5: Four iterations of Hilbert and z- filling curves [5]

2.2 DHT

In the last years a lot of work has been done by the scientists in the research of location-based peer-to-peer networks. Two main approaches have been established: the DHT-based approach and the tree-based approach.

The DHT-based overlays use existing one dimensional overlays, that are good researched, to build on top of them. Good example for such one dimensional overlay is Chord [17], that actually has only one simple function - to find the node, which is responsible for a given key. This however can be used very well to build an location based service over it. For these purposes there is a need of some linearization techniques, that can help to map the high-dimensional space onto one-dimensional space of Chord or other well known one dimensional overlays like Pastry [18] and Tapestry [19].

Probably the main technique, used to map the two-dimensional space onto one-dimensional are the space filling curves, that were already mentioned in 2.1.2. Approaches like [20] and [6] use the indexing and routing from one-dimensional overlays and apply these with the help of the space-filling curves. However it is very important to preserve the locality of the peers while mapping the high-dimensional space onto the one-dimensional space of the DHT. And in accordance to Knoll it is impossible to find a perfect mapping that is also optimally preserving the locality of the peers. [15]. Even the Hilbert curve and the Z filling curves, that we already presented and that have the best locality preservation according the study of Knoll [21], suffer from an insufficient locality preserving property.

The idea behind [6] is to create a location-based P2P system that offers query source anonymity, but is also scalable to a large number of users. The users are equipped with mobile devices e.g. mobile phone with GPS capability. The locations of the peers are indexed by an hierarchical distributed hash table, based on the Chord overlay, chosen because of its good scalability and fault-tolerance. Chord defines one-dimensional space

of index keys. Hilbert space-filling curve is used to map the two-dimensional coordinates of the user to the one-dimensional space.

In Figure 6 we can see how MobiHide works. First user U_4 starts a search for the nearest known object to his location. In our case this has to be O_4. The required anonymity has a degree K = 3. MobiHide finds out 3 random users, including the searching user, that are in sequence in the one-dimensional space. These are users U_2, U_3, U_4 and constructs a rectangle around these 3 users. Next the searching user submits a query to the location-based service, that is used. The location-based service gives to the U_4 the nearest objects of every point in the already defined square. In this example - objects O_2 and O_4. At the end measures the distance to the found objects and determines the nearest object. At the end the main advantage of MobiHide against other location-based P2P overlays is its capability to guarantees privacy. The probability of identifying the querying user is very close to the the theoretical bound. [6]

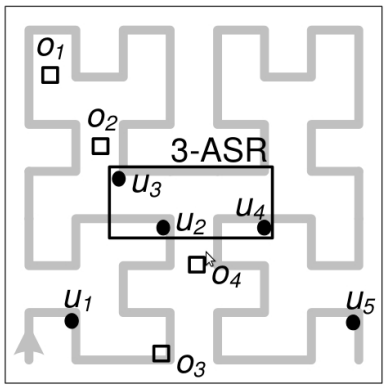


Figure 6: Anonymized query. Source:[6]

Another example for a DHT location based P2P system is GeoPeer [22]. GeoPeer is using Delaunay triangulation. The node are self-organized in it and it is augmented with long range contacts, which reflects as a reduction of the path lengths. GeoPeer, as the most of the location-aware P2P systems, can be used for geographically-scoped multicast or geographically-scoped queries.

The biggest advantage of all of the discussed approaches, based on space filling curves, is that they use already invented one-dimensional P2P architectures and build on top of them, so there is no need of developing novel system from the scratch. Unfortunately all of them have also some main disadvantages. First, because of their design characteristics, they can not provide a matching between the geographical distance of the peers and the distance in the identifier space of the overlay. This is causing additional hops in the queries and is inefficient. Second, DHTs can not fulfill requirement for completeness, as they can not provide as result all the peers in specified area.

2.3 Tree-based approaches

The second type of peer-to-peer location-based overlays are the tree-based approaches. As they do not have most of the described disadvantages of the DHT-based approaches, they are more interesting for us and we will take a deeper look at them in this section.

Probably the first overlay, that introduced high-dimensional search, even if it is based on DHT, was CAN [23]. In CAN the complete space logical and because of this it does not rely on any physical coordinate system. The space is dynamically divided between the nodes, so that every node has its own area in the universe at any point of time. The divided regions are then always equal. Every node has to manage its own routing table, which includes the IP addresses of the nodes, holding the areas, adjacent of its own. The routing in CAN is very efficient, however its complexity is preventing it of being the best solution for two dimensional space. Figure 7 shows us the partitioning in CAN.

MURK is a similar approaches, presented in [24]. Similar to CAN it is dividing the space into rectangles and every node becomes one rectangle assigned. Again this can be done with the help of KD-trees, where each node is represented as a leaf in the tree. The biggest difference between CAN and MURK is that CAN is partitioning the space equally. MURK however splits the load equally. This should provide better performance, which is the bottleneck of CAN.

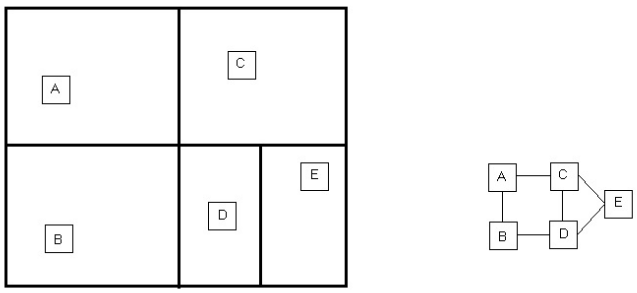


Figure 7: CAN - Content Addressable Network - each node is assigned to a rectangular area and each node maintains connection to its neighboring areas

There are also solutions, based on the Quad trees, like [25]. In this work Waresiak and Skrzynski propose to split the universe into subspaces and then leaf nodes of the tree, which represent the smallest subspace has to be assigned to the peers. At the end the search is performed by a graph created only by the leaf nodes of the tree.

Next we will present the SGR-Tree [7], a R-tree, developed on the top of the Skip Graph [26], which is another P2P overlay network. Each Skip Graph node reflects in a R-tree leaf node. The whole space is virtually divided into non-overlapping subspaces at each Skip Graph node. Each of these subspaces is connected to the node trough a neighbor node. This helps, when a node receives a query. In this case, it just forwards the query to the right neighbor node. The right neighbor is chosen by determining the minimum covering

hyper-rectangle for each region and finding the node, whose minimum hyper-rectangle intersects with the search region. Thus there is no root node, which can be overloaded by the queries and also the high cost for managing the R-tree nodes is avoided. Figure 8 shows an example of a SGR-Tree.

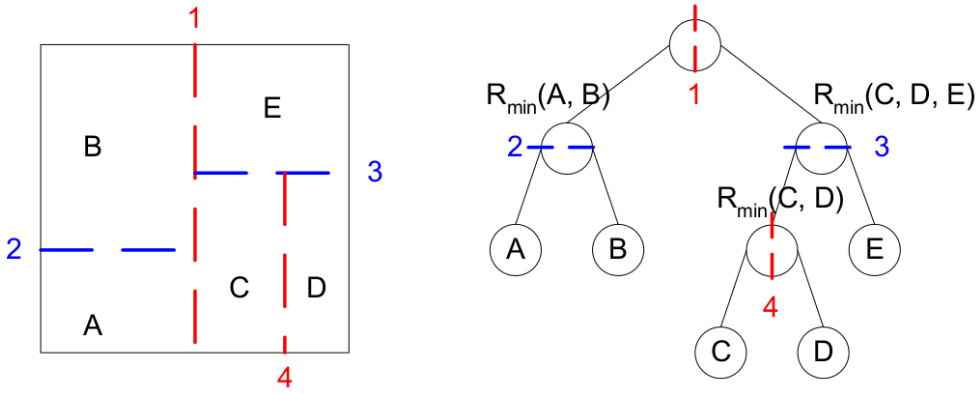


Figure 8: Example of a SGR-TREE. Source:[7]

Another R-tree based overlay is the P2PR-tree [27]. As opposed to the SGR-Tree the nodes and the leaves of the tree do not have the same role. In the first two levels of the tree, the space is statically divided into blocks at the first level, which are divided into sets of rectangular ties, called groups, at the second level. At the third level, each of the groups is dynamically divided into further subspaces, depending on the number of the peers inside it. Unfortunately the subspaces may overlap, causing expensive and poor performing searches in location-based services. Another issue of the [27] is that it does not describe how a new peer has to join the tree, if it is not falling inside one of the dynamically created subspaces.

There are also works, based on the KD-trees as [28] and [29]. Depending on the number of objects in the overlay. SkipIndex hierarchically divides the spaces into subspaces using KD-tree. After that a one-dimensional key is created for each region. The key captures the position of the region in the KD-tree. At the end all the keys are used to store the leaf regions in a Skip Graph.

Another interesting work is EZSearch [30]. As opposite to the other solutions, that we have already presented, it is using the ZigZag hierarchy which is a hierarchy with several layers of clusters. The size of the cluster may vary, but is defined by parameter z. It can be between z and $3z$. At the first layer all the nodes are organized in clusters with size z. For every cluster there is a head node and associate-head node. All the head nodes are then presented in the second layer, if the number of these nodes exceeds the maximum cluster size of $3z$, they are partitioned again into cluster with maximum size of $3z$. This has to be done, until there is a layer with number of nodes, that does not exceed $3z$.

A very interesting hybrid solution is presented by Kovacevic in [8]. Globase.KOM relies on super-peers to maintenance the hierarchy of the areas. It is a clustering-based solution. This is giving it a great flexibility of space dividing in case of non-uniform distribution of

the peers. Unfortunately Globase.KOM does not exactly fulfill the requirement for scalability as a special role in the area hierarchy creation is assigned to the some super-peers. These can get overloaded or even worse - they can fail, which can cause partitioning of a large part of the network. We will take a more detailed look at the Globase.KOM in the next section.

The last overlay that we will present is GeoP2P, proposed by Asaduzzaman and v.Bochmann in [9]. Even if it is inspired by Globase.KOM, it has one main difference. All the peers in GeoP2P have equal roles, which makes the overlay more scalable. This is also the main reason to use it as basis for our work. At the next sections we will take a deeper look at GeoP2P, comparing it to Globase.KOM.

2.4 Globase.KOM

At this Section we will take a deeper look at Globase.KOM, as it is one of the best evaluated overlays and thus it is giving us a good basis to compare with. We will present the design, the routing and maintenance strategies of the overlay and will compare them with GeoP2P. At the end we will make a short overview of the fulfillment of the requirements, introduced in 1.3.

2.4.1 Overview

Peers in Globase.KOM area heterogeneous - they have different network connection and CPU resources. Because of their heterogeneity we have two kinds of peers in Globase.KOM - normal peers and superpeers. Thus the overlay is hybrid. Superpeers are connected in the overlay, building a tree structure. The universe is presented as a rectangular area, which is divided in smaller non-overlapping subareas. Every superpeer has an subarea assigned to it and if the subarea of one peer contains the subarea of another, then the first superpeer is called parent of the second. The basic structure of Globase.KOM is shown in Figure 9.

The superpeers are also managing connections to the superpeers of other branches or to the superpeers having higher level in the same part of the tree. A load threshold is defined and if the number of peers inside an area exceeds this, a new subarea is created. This new subarea is then assigned to the peer with the best bandwidth and the most resources, making it a superpeer of this subarea. When a peer starts an area search for selected subarea, the query is redirected to the superpeer, which this subarea is assigned to. In case of missing intersection of a area in the query with the area of the superpeer, the superpeer is forwarding the query to his parent. This has to be done, until intersection is present.

This basic topology is already known from other overlays, however in the case of Globase.KOM it is extended by interconnections. All superpeers, except of the superpeer in the root area and the superpeers in the leaf areas, manage connections to their

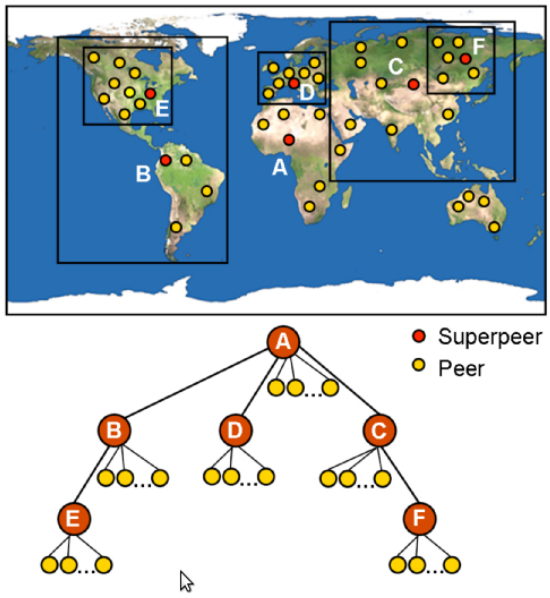


Figure 9: Globase.KOM overlay. Source:[8]

parent and their children superpeers. Beside these connection, they manage connection to superpeers from other branches, called interconnections. This should reflect in a better load balance and decreased hop count, as most of the cases it prevents from routing trough the higher-levels of the tree.

The interconnections are divided into two types - high interconnections and wide interconnections. The high interconnections are these between the superpeers of the same branch and are used to avoid failures and to recover after these. The wide interconnections has the assignment to prevent routing in higher levels of the tree and thus are improving the routing performance and providing better load balance. Wide interconnections are the interconnections between superpeers of different branches. There are two ways of receiving interconnections. First they can be obtained from the parent superpeer or they can be piggybacked on the communication messages.

Further to prevent failures, all the routing table information is periodically sent from every superpeer to another peer in the same area, which fulfill the requirements for being superpeer of the area. In case of failure of the superpeer, this second peer is getting superpeer of the area.

2.4.2 Design decisions

At this Section we will take a short look of some design solutions like joining the network, forming the subareas and solving the overlapping problem, as they are described by the author.

Joining the network - Joining the network is a crucial part of the peer-to-peer networks and so bootstrapping is very important for building the network. The author describes the joining method of Globase.KOM as mediator approach. Other bootstrapping methods are described in [31]. In the mediator approach, a server is used to save the contact information of the peers, used for the bootstrapping. If a new peer wants to join the network, it has to contact the server and after getting the list of contact peers, it is chooses one of them and send its geographical coordinates to it. The contacted peer uses the routing strategy, described in 2.4.3 to find the superpeer, that is responsible for the area, the peer belongs in. The bootstrapping peer is sending a message to this superpeer. The superpeer sends a message with its interconnections and the contact information for the new peer and thus the join of the new peers is finished.

Forming of the areas - Another important design decision is the forming of the areas. We already explained that the universe is presented by a rectangular area. This is also the first area in the overlay. Once the first peer joins, this first area is build and is assigned to the first peer. Much more interesting is the forming of the next areas. The algorithm presented by the author, decides when to create new area, by measuring the load of the superpeer. The load is presented in this case by the number of peers inside the area, the superpeer is responsible for. The load is divided by two thresholds L_1 and L_2 into three levels. The first one is the normal load, the second one between L_1 and L_2 is when the area is overloaded, and if the number of peers exceeds L_2, a new area inside the own area is created. It has to be rectangular, can not overlap with other areas and its number of peer has to be around L_2 - L_1.

Resolving overlapping - The strategy for forming the areas has one disadvantage - it can cause overlapping of the areas, so the author introduces an algorithm for preventing the overlapping of the areas. It has three main parts - building a hotspot, resolving the overlapping and extending the area. First a area of the universe with high concentration of peers, called hotspot has to be found. For these purposes a single linkage clustering algorithm, introduced in [32], is used. Once the hotspot is build, it is checked for intersections against the other areas inside the area of the superpeer. In case of intersection peers are removed from the hotspot, until it does not intersect with any other areas. At the third step, the hotspot is extended with new peers until it reaches number of peers approximately L_2 - L_1.

2.4.3 Routing strategy

We will consider three different type of queries and the routing strategies needed for them. According to the author of Globase.KOM the main goal of the overlay is to find the peers that are:

- inside selected area
- or are at specific location
- or are closest to specific location

The author describes the routing strategy of the overlay as greedy, "meaning that the

query is forwarded to the neighbor which is closest to the destination." [8] First a check in the connections of the peer, that starts the query, is made. After that a lookup message is sent to a suitable interconnection and if there is no appropriate interconnection, the lookup message is sent to the higher levels of the tree, using the parent connections. This iteration has to be done until appropriate location and the correspondent peer is found.

The strategy for finding peers inside selected area is very similar. The superpeers are again sending query messages trough their interconnections or parent connections. The main difference is, that every time when a superpeers receives a query message, it checks if the searched area intersects with its own area. If this is true, it is sending the peers inside his area to the peer that has started the query. As already mentioned a list of interconnection is sent with every query message, so the interconnections are dynamically updated all the time.

The last routing case mentioned by the author is finding the closest peer. First the peer that starts the query is checking his known contacts and trough distance calculation is determining the closest one. After that it is sending query message to the superpeer, responsible for the area of the closest peer, found in the distance determination. This superpeer is checking if there is a closer peer outside its area. If such a peer is found, the query message is sent to the parent superpeer, who is then responsible for finding the right superpeer.

2.4.4 Maintenance strategy

The maintenance strategy of the overlay is divided by the author into two parts - failure detection and its recovery. Further the maintenance depends of the different kinds of connection inside the overlay. Here we will present the failure detection and failure recovery for the most crucial connections, as they are described by the author.

The connection between the superpeers is periodically checked by sending messages. If three of these messages in a row do not receive an answer, a failure is detected. The information about the failure is then sent down to the three, resulting in providing this information also to the regular peers. The frequency of sending these messages is higher for the superpeers that have direct parent-child connection, as this is crucial for the tree structure.

There is a similar situation between the peers and their superpeer. The regular peers are sending check messages to their superpeer and if they do not receive answers to three messages in a row, a failure is detected. The superpeer at the other also receives alive messages from the regular peers in its area. This is however not so crucial for the tree structure and the frequency of these messages is not so high.

The connection between regular peer and its interconnections peer is not checked actively, as this information is provided to the peers by their superpeer. The same goes for connection between peer and the root peer. This information is sent down to the tree to all the peers and thus the overload, caused by so many messages, is not necessary.

If a failure is detected, its recovery has to be started. This is preserving the tree structure and ensures the routing will work regularly. The failure recovery methods depend again of the connection type.

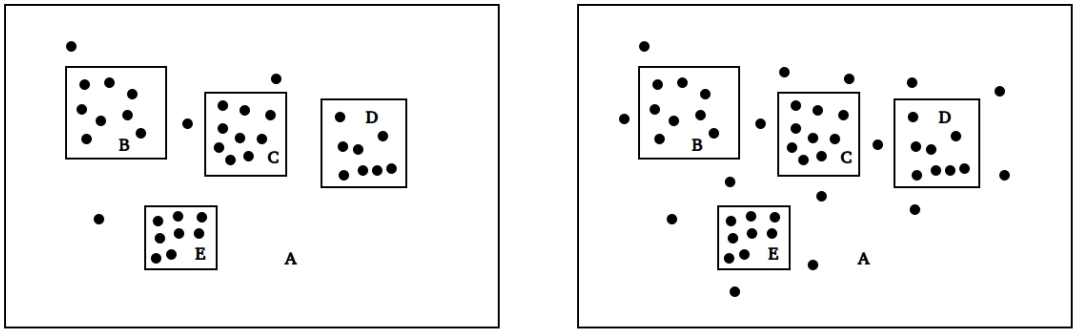
As already mentioned a replica peer is chosen and the routing table of the superpeer is periodically sent to it. In case of failure of the superpeer, this second peer is getting superpeer of the area. The closest peer, that fulfills the requirements for superpeer, is chosen as replica peer. If a failure of the replica peer is detected, the superpeer chooses another replica peer. If both the superpeer and its replica peer fail, a regular peer inside the area uses the interconnections to find out the parent of the failed superpeers. The parent superpeer sets the peer that sent the message as a new superpeer of the area.

If a regular peer inside an area fails, the superpeer just detects this failure, but still preserves it in his list of peers. This is done, because most of the peers are coming back after certain time. As mobile peers are irrelevant for Globase.KOM, the peer will appear at the same location next time, when it joins the network again. If a failure of an interconnected superpeer is detected, the peer just removes it from its list of interconnections. Further the interconnections are updated with the next messages.

In the most interesting case of failure of the root superpeer and also of its replica, a superpeer of the next level of the tree is taking over the root area. However this is tricky, as it can cause several independent trees. Because of this the author introduces an election algorithm for choosing the new root superpeer.

2.4.5 The "missing sheep" problem

Even if the single-linkage clustering algorithm is well known, during our work on this Thesis, one main problem was found. To introduce this problem, that we call the "missing sheep" problem, we will consider the following scenario: We have our main rectangle area A with peers inside of it. To describe the problem, we will assume that the peers are sheep. So the shepherd decides that it will be useful to put the sheep in a sheep pens - these are the underareas of area A, if their number exceeds 14 - this is our L_2 threshold. The L_1 threshold is 6. So every sheep pen has to have a size of the difference between L_1 and L_2, which is 8. In Figure 10a the area A is already divided into some subareas, because the number of peers has exceeded the L_2. These are the subareas B, C, D and E. In Figure 10b new peers have joined the network and at a specific time point the number of the peers in area A has raised up to 15. This one more than the threshold L_2. So the dividing algorithm is started. First a hotspot is created, but because of the positions of the new peers, it is overlapping with all the subareas B, C, D and E. However it is impossible to shrink the hotspot in such manner, that a creation of a new subarea with size at least L_2-L_1 is possible.



(a) (b)

Figure 10: The "missing sheep" problem

2.5 GeoP2P

The second location-based overlay, that we will take a deeper look at, is GeoP2P, proposed by Asaduzzaman and v. Bochmann in [9]. It is inspired by Globase.KOM, but has also some keys differences. Unfortunately there is no evaluation presented of this overlay, nevertheless comparison of the design, the routing and maintenance strategies of GeoP2P with these of Globase.KOM is possible. This will be done in the following sections.

2.5.1 Overview

Similar to Globase.KOM, GeoP2P is an hierarchically structured overlay. The peers in the overlay can be heterogeneous, however according to the authors, they all have the same role. This is the first main difference with the work of Kovacevic. The universe is again presented as a rectangular area, which is later divided into smaller subareas. At this point we have the second main difference and this are the methods of division. In GeoP2P there are two methods of division. The first one is the well known from Globase.KOM clustering. The second one is splitting. When splitting is used, every area is split into smaller areas at the x or y axis, depending on the length of the axises. As already mentioned, all the peers in GeoP2P have the same role in the overlay. Opposite to Globase.KOM, the area is not assigned to a specific peer. All peers in the area are equal and every peer knows the other in the area. Thus a node in the tree structure of the overlay is representing an area and not a peer. An example of a GeoP2P overlay, created by clustering and the correspondent tree are shown in Figure 11.

Every peer in the overlay manages their own routing table. The routing table contains the contacts, that the peer knows. The peer knows all the peers in a selected area or it knows some peers that have further information about this area. If a query is started, the peer checks if it knows another peer, that belongs to an area intersecting with the searched area. If no such peer is found, the query is forwarded to a peer with better

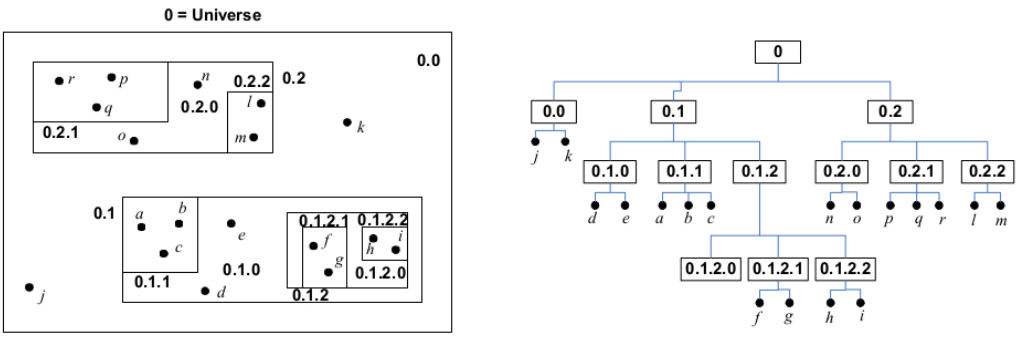


Figure 11: GeoP2P overlay. Source:[9]

information about the searched area. Another key difference to Globase.KOM are the missing interconnections.

2.5.2 Design decisions

In this Section we will take a look at the most important design decisions, similar to 2.4.2.

Joining the network - The authors assume that before joining the network the peer knows his own geographical coordinates, network address and the contact information of at least one peer in the network. Unfortunately the authors do not mention how the peers are going to obtain the contact information of peers, that are already in the network. At this place and also in our work in the next Chapters, we will assume this information as given. The first step of joining the network is to find a peer in the area, that it will belong to after joining. To do this, the peer that wants to join sends a message to one of the known contacts. The contact checks if the asking peer has to be in its own area. If not, it uses its routing table to find a peer in the right area and sends the message further to this peer. The peer in the right area then answers to the asking peer with a list of all peers in its own area and also a copy of the routing table. Meanwhile it tells the other peers in the area, that a new peer will join the area. Once the peer joins the network, the information of the join process is provided to peers in other areas in the network by piggybacking the routing tables of the peers in the area on other messages.

Forming of the areas - The areas in GeoP2P are formed in similar manner to Globase.KOM, however there are also some differences. The first area again is a representation of the universe. After that this area is divided into subareas, forming a tree structure. The areas are dynamically divided into subareas using two thresholds again. They mark the number of peers in the area. And at this place there is the first big difference with Globase.KOM. There is also a parameter k settled. This is used to determine how many subareas can be an area divided into. So if k is 3, so an area can be divided into three subareas. These 3 children area completely cover the parent area. The children areas are also always not-overlapping. The second main difference is that the authors

propose two methods of division. The first one is the clustering, already known from Globase.KOM. There is no information about the used clustering algorithm. However the "missing sheep" problem will be probably present again if the authors use the clustering method.The second method of division is splitting. Here every rectangular area is divided into k subareas, which areas are also rectangular. The X-axis and the Y-Axis of the divided area have to be compared. In case of equality the area is divided along the X-Axis. Otherwise the area is divided along the longer axis. In Figure 12 a simple example of the splitting method of division is shown.

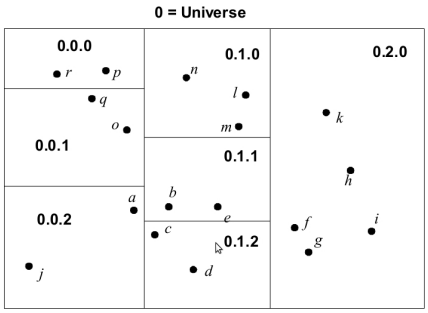


Figure 12: Splitting method in GeoP2P. Source:[9]

Resolving overlapping - The authors do not provide information about preventing and resolving overlapping of the areas in the clustering method. In the splitting method overlapping is not possible, as the areas are divided along one of the axises.

2.5.3 Routing strategy

The authors of GeoP2P present three types of queries, that are similar to the already introduced in Globase.KOM. These are again queries targeted to all peers in the same area, queries targeted to just one peer at a specific location and queries to a peer, that is closest to a point in the universe.

First if a message has to be delivered to all peers in the same area, the sender checks its routing table and sends the message to all peers, which coordinates intersect with the desired area. They check their area and resend the message to all the peers in their own area, which intersect with the asked area. This can also be used for update operations or other services. During its way to the target peers, a parameter $level$ is defined in the message. This is correspondent to the level of the tree and is preventing sending of the message to the same or higher level of the tree and also the message is delivered only once to the searched peers.

The second type of queries is a query to a peer or point at a specific location. Despite this method is similar to the previous, there are also some differences. First a peer sends the message with the coordinates of the point to the peers, which have an area intersecting with the point. These peers check their own areas and if the point is not inside it, then the

message is re-sent to the next peers, having an intersecting area down on the tree. This has to be done, until a leaf area intersecting with the point is found. As the point can not intersect with several areas, only one leaf area is found.

Finding the nearest peer to a selected point is a little bit more tricky. First the area, where the point belongs, is found, using the previous algorithm. Once this is done, the nearest peer inside this area is found. However it is also possible that another peer, outside this area is the nearest. So the nearest peer in the found area determines its distance to the searched point and starts an area search, where the area is a circle with center the searched point and radius the determined distance. In this way also the nearest peers in the neighbor areas are found and then the distances of all of them are compared.

2.5.4 Maintenance strategy

As there are no different roles of the peers and no interconnections in GeoP2P, the maintenance has to be much more simple. The authors do not provide exact algorithm for detecting and recovery of failures. They assume that periodical updates of the routing table are done and also some bucket of peers are saved for every area, instead of one peer. Similar to Kademlia [33] the peers have a rating, according to their time of activity. Peers that are longer active, are pushed forward in the list.

2.6 Conclusion

In this Chapter we have presented the two methods for spatial data indexing, with focus on the space partitioning in 2.1.1, as it is the most used method in location-based overlays. Also the space filling curves were presented in 2.1.2, as there are also some approaches based on them. Furthermore some location-based overlays were presented and divided into two categories, depending on the used approach. The two main approaches, DHT- and tree-based approaches, are presented respectively in 2.2 and 2.3.

We took a deeper look at two location-based overlays. First Globase.KOM [8] from Kovacevic was presented in 2.4. We decided to use it, because of its good evaluation will help us to make a real comparison with our solution in the next Chapter. Further the GeoP2P location-based overlay [9] was presented, as our solution will use it as basis. Unfortunately both of these overlays do not consider movement of the peers and thus a further comparison in case of movement will be impossible. Another common problem of these two overlays is the "missing sheep" problem, that we already described in 2.4.5.

In Chapter 3 we will present GeoP2P.Extended - a novel motion-aware location-based peer-to-peer overlay. In Section 3.1 a brief overview of the overlay will be introduced. Further in Section 3.2 the design decisions, including the joining the network, the forming the areas, the routing and the maintenance strategies, will be examined. At the end, in Section 3.4.2, the main difference to the Globase.KOM and GeoP2P - the movement algorithm, will be described.

3 GeoP2P.Extended

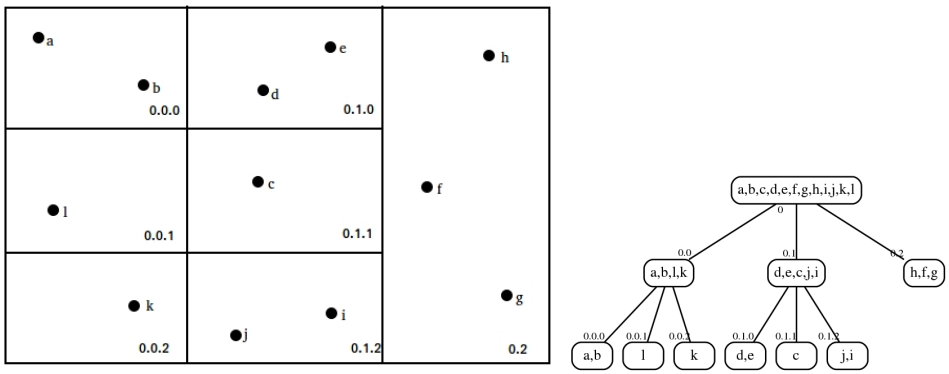
In the previous chapter, the main methods for spatial data indexing were presented. Furthermore some of the related work, done by the researchers in the last years in the area of the location-based peer-to-peer networks, was introduced. All the approaches were divided in two groups - DHT- and tree-based approaches. Considering the algorithms and methods, presented by Kovacevic in [8] and Asaduzzaman and v.Bochmann in [9], the design, implementation and evaluation of a novel location-based peer-to-peer overlay is presented in this Thesis. This novel overlay, called GeoP2P.Extended will be introduced in this chapter. In Section 3.1 we will make a short overview of the overlay. The design decisions, used in the overlay, will be introduced in 3.2. Further in 3.3 and 3.4.1 the routing and the maintenance strategies will be presented. As our overlay is considering movement of the peers, the movement algorithm will be presented in 3.4.2.

Globase.KOM was chosen, because of its good implementation. Also a good work has been done by the author in the evaluation and describing the design of the overlay, making it suitable for a future comparison with our new solution. Furthermore well known ideas like the tree structure and the single-linkage clustering algorithm are used in the design of the overlay. Unfortunately there are also some design decisions, making Globase.KOM not suitable as only point of comparison. The hybrid character of the overlay, including superpeers and normal peers, introduce a big difference to our use case, where the all the animals and researchers have to have equal roles in the network. Other disadvantages are the missing consideration of movement of the peers and also the division of the areas by clustering, causing the "missing sheep" problem, that we have introduced in 2.4.5 . Thus there is a need of another location-based peer-to-peer overlay to compare with. For this purpose the GeoP2P has been chosen. It considers equal roles of the peers in the network, making it more suitable for comparison with our solution. Furthermore the authors introduce two methods of division - clustering and splitting and even if they concentrate on the clustering algorithm in their work, there is also the possibility to use the splitting method and thus to avoid the "missing sheep" problem. Because of its features, GeoP2P has been chosen as basis of our novel location-based peer-to-peer overlay.

3.1 Overview

Even if peers in GeoP2P.Extended can have different bandwidth, CPU power or RAM, all of them have equal roles. Thus the overlay is not hybrid like Globase.KOM. The universe is again presented by a rectangular area, which is divided into smaller not overlapping subareas. When an area is divided into subareas, the subareas are getting children of the divided area and the divided area is getting parent of the subareas, thus building a tree structure. Example of the GeoP2P.Extended overlay is shown in Figure 13a. In Figure 13b the corresponding tree structure is shown.

Areas are divided into k equal subareas, where k is a predefined parameter. The division is done, when the number of peers exceeds a predefined threshold. Every peer has its



(a) (b)

Figure 13: GeoP2P.Extended - example of division and corresponding tree

own area, which is the leaf area in the tree, which the peer belongs to. However one leaf area can be the own area of several peers. In fact every leaf area is the own area of all the peers inside it. Every peer saves a list of all peers inside its own area. All other areas, known by the peer, are saved in a partial tree, called *own tree* of the peer, which is actually the own view over the global tree. This own tree has also the function of a routing table of the peer.

The peers have information only about the division of their own area. Thus after every division of the own area, a new level is added to the own view of the tree. As there is a new leaf area, which the peer belongs to, this area is assigned as new own area of the peer. The other subareas, which the old own area was divided into, will be called "neighbor areas" in this work. The peers are not considering the divisions of the neighbor areas. Considering the division shown in Figure 13a we can see the own view over the global tree of peer c in Figure 14.

As we can see, the information about the division of area 0.0 is missing in the own view over the tree, because peer c does not belong to area 0.0 and thus it does not consider the division of this area.

Every peer manages connection to all the peers inside its own area and also to the peers of every neighbor areas at every level. The number of peers in the neighbor areas, which the peer has connections to, is defined by parameter. The connections to the peer from the neighbor area are very important for the routing strategy, that will be presented in Section 3.3 and thus the information about the peers in the neighbor areas is periodically updated.

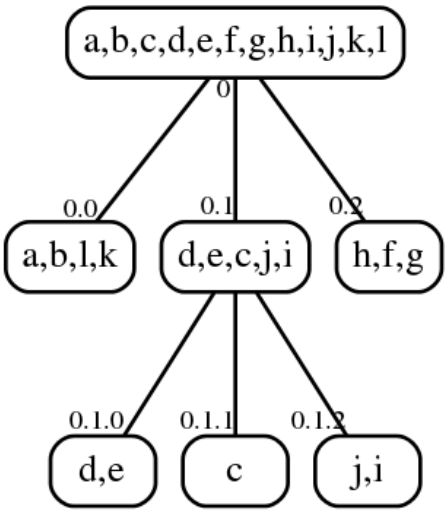


Figure 14: Own view over tree of peer c

3.2 Design decisions

In this Section the design of the novel overlay will be introduced. In 3.2.1 the joining method of the overlay is presented. As GeoP2P is used as basis for our overlay, the idea of the splitting of the areas is adopted and extended in GeoP2P.Extended. Further information about the splitting of the areas is provided in 3.2.2. As the splitting method has been used in our overlay, there is no need of a method for resolving the overlapping.

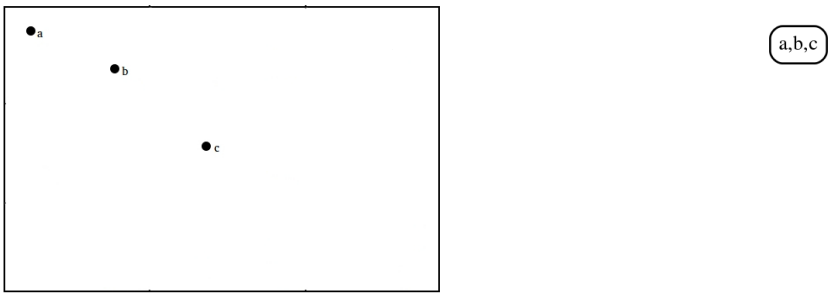
3.2.1 Joining the network

For joining the network, the mediator approach, presented in [8] will be adopted and implemented in our overlay. The mediator method is also chosen, because of its simple implementation in the simulator that we have used for our work and that is introduced in Chapter 4. Again there is a list with all the nodes and their contact information on a server. When peer wants to join the network, it asks the server for a contact peer and sends its own contact information and geographical coordinates to it. The contact peer checks if the new peer has to be in its own area and if not is resending the message to the peer that is closest to the position of the new peer. This has to be done, until the right leaf area for the new peer is found. Once the right own area for the new peer is found, a peer of this area sends its own view of the global tree to it. Then the peer sends its information to all the peers inside its own area, which have the task to add it to their list with peers in the own area.

3.2.2 Forming the areas

Forming of the areas is another very important design decisions, when creating a location-based peer-to-peer overlay. In our work the splitting method, proposed in [9] will be used, but also extended for our needs. When peer joins the network, a new rectangular area, representing the universe, is created. This area is also assigned as *own area* of the first peer. Again a threshold is predefined and it is representing the maximum number of peers inside an area, before it has to be split. When the number of peers inside an area exceeds the predefined threshold, the area is split in k rectangular subareas, which have the same size. This is the main difference to the splitting method in GeoP2P, where the subareas have to have similar number of peers inside them. We are not using this method, because we are considering future movement of the peers and as their new position in the future can not be predicted, the load of area is continuously changing over the time. So the right size in the future, considering the current load, can not be predicted. This is also the reason of having only one threshold, as merging of subareas would not be part of this work. When splitting, the X-axis and the Y-axis are compared again. The area is then split along the longer axis. In case of equality of the axises, the area is split along the X-axis. While splitting, every area is assigned to an ID. The first area, representing the universe, is assigned to ID 0. After every splitting, the ID of the split area is taken and a continuous number is added to it to become the ID of every subarea.

In Figure 15 the universe before forming of the new areas inside it is shown. First peers a, b and c join the network and form the first area with ID=0, which is also representing the universe. The area and the corresponding tree are shown in Figure 15a and Figure 15b. Further in Figure 16 an example of the forming of the areas is presented. We assume that every area has to be spit when the number of peers inside it exceeds 3. Every area has to be split in k=3 equal, rectangular areas. So when peer d joins the network, already presented in Figure 15, the number of peers exceeds 3 and thus the universe is split into subareas 0.0,0.1 and 0.2. Further peers e,f,g and h join the network without exceeding the threshold. The universe and the corresponding tree area shown in Figure 16a and 16b. In Figure 16c and 16d area 0.1 is split into 3 subareas, as result of the join of several new peers. At the end , in Figure 16e and Figure 16f area 0.0 is also split into the subareas $0.0.0$,$0.0.1$ and $0.0.2$.



(a) (b)

Figure 15: GeoP2P.Extended - example of division and corresponding tree

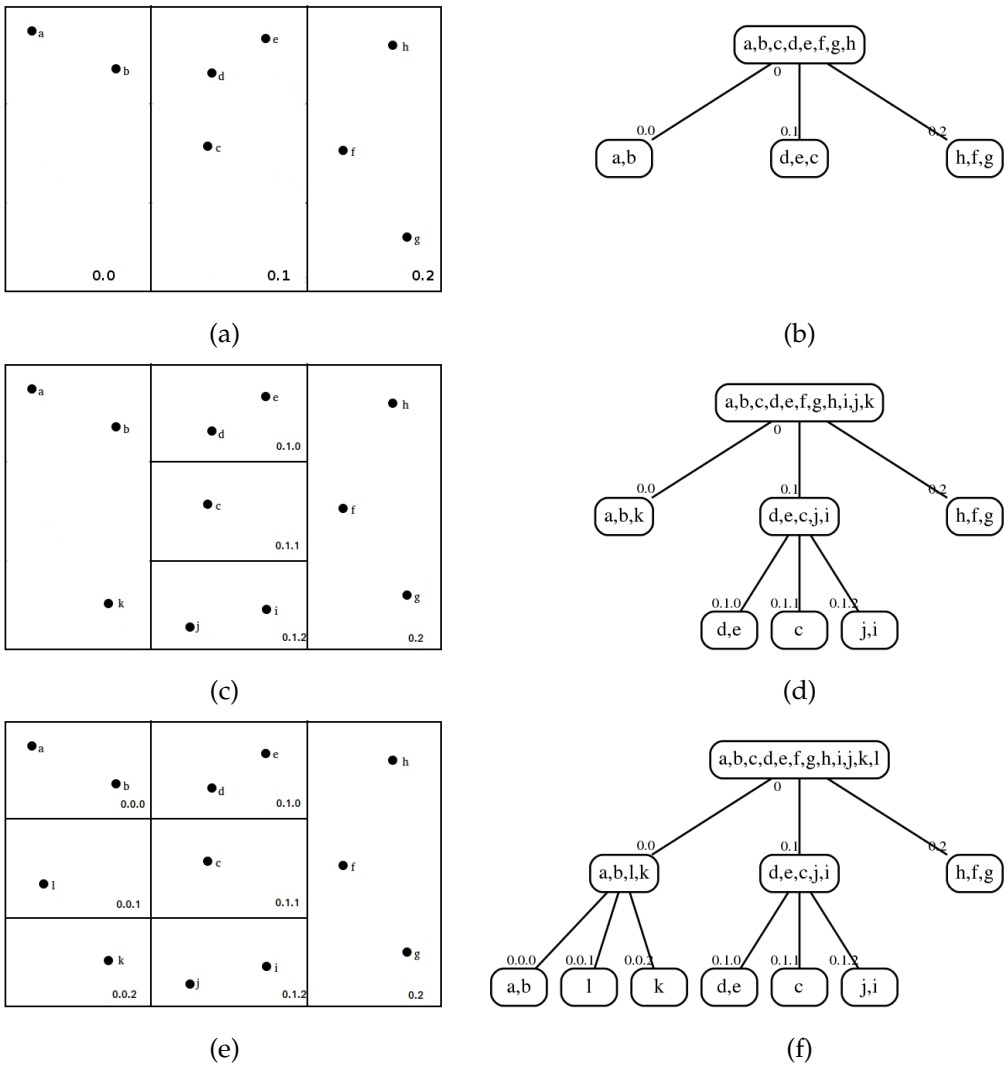


Figure 16: GeoP2P.Extended - example of the forming of the areas

3.3 Routing strategy

Similar to Globase.KOM and GeoP2P , three types of queries can be defined:
- queries to a selected point in the overlay
- queries to all peers in a specific area
- queries to the k peers, that are closest to a specific point in the universe

When a specific point in the universe is searched, e.g. when peer want to join the networks and is searching for its area, the asking peer sends a query message to random peer. This random peer checks its own area for intersection with the searched coordinates. If the coordinates are inside the own area, the asked peer answers to the asking peer with its own area and the right point. If the point does not have intersection with the own area, then the peer checks its neighbor areas, which has the same level as the own area. In case there is no intersection with the neighbor areas of the same level as the

own area, the asked peer checks the neighbors at the next higher level. This is done until a neighbour area is found, that intersects with the point. When such a neighbour area is found, the query is resent to a random peer of the intersecting neighbour area, which then repeats the same procedure. This is done until the right area is found - Listing 1

Listing 1: GeoP2P.Extended routing strategy

```
if contact !=null
        send RequestRoutingTableMessage(contact)

else
        if askingContact is in Area(ownArea)
                increase successful lookups
                send RequestResponce(ownArea, ownTree)
        else
                rightArea = findClosestKnownArea(asking.x, asking.y)
                randomContact = random of rightArea.Contacts
                send RequestRoutingTableMessage(randomContact)
                increaseHopCount
```

In the second case, when the message has to be sent to all peers in a specific area, a similar approach is used. The asking peer first checks the ID of its own area. If the own area has the same ID as the ID of the searched area, then the right area is found. If the own area is not the right area, a mirror approach is used. In this case the asking peer starts the search from the top of the tree. First it check the children of the root area. The asking peers compares their ID with the ID of the searched area. When a child of the root area is found, such that the searched area ID starts with this child's ID, then the search goes in the next level of the tree an this child area is saved as current right area. If the searched area ID starts with the current right area ID, but it is not the same, then the asking peer checks if it has information about the children of the current right area. If it knows the IDs of these children areas, the same procedure is repeated. If the asking peer does not have this information, it resends the query to a random known peer of the current right area. This is done until an area with the same ID like the searched area is found. When this is done, the query is re-sent to a known peer inside this area, which then answers to the answering peer with his full list of contacts in the own area.

In the last case - queries to the closest peer, the same approach as in Globase.KOM is used and thus it would not be further discussed in this work.

3.4 Movement and maintenance strategy

In the previous sections we presented the design decisions and the routing strategy in the same manner as we did for Globase.KOM and GeoP2P. As both of them do not consider movement of the peers and at the other side our solution is motion-aware location-based overlay, there will be some differences in this section. First we will present the maintenance strategy as we did for the two other overlays, but in the second part of the Section the movement strategy of GeoP2P.Extended will be presented.

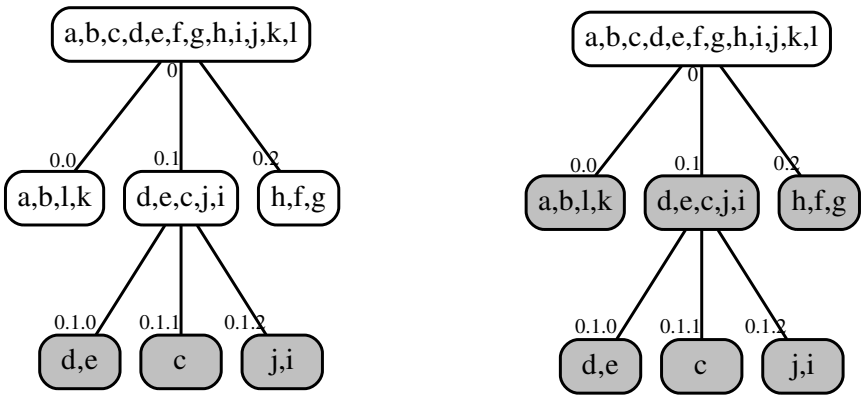
3.4.1 Maintenance

As there are no superpeers in our overlay, as this is the case in Globase.KOM, maintenance strategy is a little bit different. First, failing of individual peers is not so crucial, as it will be if a superpeer fails in Globase.KOM. So in our overlay, we decided to use two kinds of updates for maintenance of the network. The first way of updating the information is the piggybacking of the update information on the top of the query messages. For example if peer A sends a query message to peer B, peer B answers to peer A with the actual information about its own area.

The second way to update the information is the periodic update. Here there are three predefined parameters, used for the updates. First we have the parameter t, which is the interval of time for doing updates. If t=20 seconds, then every peer sends update messages every 20 seconds. The second parameter, and this one is more interesting, is l, which is the level of update. If a peer decides to make update of his own view of the routing tree, it sends messages to the other peers in its own area and to a random peer in the other known areas. And here is the role of the parameter l. If l=0, the peer sends update messages only to the other peers of its own area and to a random peer of the neighbour areas of its own area. If l=1 the peer sends messages also to the neighbour areas in the higher level. Adjusting the parameter l, the use in different scenarios is possible. In Figure 17 an example of updates of different levels is shown. In Figure 17a the parameter l=0 and thus only the own area and the neighbour areas of the same level are updated. In Figure 17b, where l=1, also the neighbour areas at the next higher level area updated. In case of failure of the peer, that was chosen to receive the update message, another peer from the same area has to be chosen. When the asked peers receives an update message, it answers with the current peers in the searched area. Because the information about the own area has always to be up to date, the peers in the own area are update much more often. This is the reason for introducing last parameter - s. It represents the number of iterations of updates of the own area, that has to be done, before an update of other areas is possible.

3.4.2 Movement

In opposite to the other two overlays, the peers in GeoP2P.Extended can change their position in the universe. For these purposes there are some main steps, that have to be done, if a peer wants to move over the universe, which will be described below.
Inform the neighbours - the first step is to inform the contacts of the own area, about the leaving it. Also at least the neighbours on the same level as the own area has to be informed. Again the parameter l to define the neighbours at how many levels above the own area has to be informed.
Search for the right area - at the second step, the moving peer starts the routing operation, that we already described in 3.3, in order to find its future own area. The only difference is that at this step the bootstrapping list is not used. The peer makes looks at his own view of the global tree and finds a peer, that is closest to the new coordinates of the moving peer. The query message is then sent to this peer.



(a) (b)

Figure 17: Update of different levels in GeoP2P.Extended

Join the new area - at the third step the moving peer receives its new own area, joins the new area and informs all the other peers in this area about the join.
In out analysis of the movement algorithm we found out, that if a lot of peers move to another areas, this can impact the routing strategy , that we presented. Thus a new structure has been added - the list of the old own areas. Every time when a peer moves to another area in the universe, the old own area is saved in this list. The list has to be updated together with the other areas, to ensure actual and proper information about the old own areas. So if the peer becomes a query message from another peer, that does not know about the movement of the asked peer and still has the information, that the asked peer is in its old own area, then the asked peer can provide to the asking peer actual information about the searched area. In order to reduce the network load, the number of old own areas has to be limited by a parameter.

3.5 Conclusion

In this Chapter we introduced our novel location-based peer-to-peer overlay - GeoP2P.Extended. Similar to our overview of Globase.KOM and GeoP2P, the method for joining the network has been described in 3.2.1. The well know mediator method is used to provide a list of contact in the network to other peer, that want to join. In Section 3.2.2 a method similar to the splitting method from GeoP2P is used to divide the areas in equal, non overlapping subareas.

At Section 3.3 the routing strategy and different type of queries area introduced. The own view over the global tree has main role in all type of queries. Depending on the query, the search over the tree start from the top or from the bottom. Further in Section 3.4.1 the update strategy has been introduced as this is crucial for the proper routing over the network. In Section 3.4.2, the main difference to the other two overlay is presented - the

movement algorithm. In order to move to another area in the universe, every peer has to inform its neighbours, then to search the right area and to join it. At the end the old own area is saved to a list, needed for the routing.

In Chapter 4 we will take a look at the simulator, that has been used for the evaluation of our overlay. Further the implementation of the tree structure, the routing and update strategies will be presented.

4 Implementation

Until now the main methods and techniques, used in location-based peer-to-peer overlays, have been presented. Also a short comparison between two very interesting overlays - Globase.KOM [8] and GeoP2P [9], and also our novel location-based peer-to-peer overlay - GeoP2P.Extended, was made. In this Section we will take a look at the implementation of GeoP2P.Extended. In 4.1 we will present the simulator, that we have used for the simulation of our overlay. An overview of the implementation of the overlay will be given in 4.2 . Since the beginning of the century a lot of peer-to-peer networks simulators were developed. P2PSim [34] and Peersim [35] are some examples for peer-to-peer networks simulators. These simulators however have some disadvantages, like missing support for dynamic networks and poor performance, making them not suitable for a simulation of location-aware peer-to-peer network, which is also considering movement of the peers. Another simulator, PeerfactSim.KOM [36] is offering us the needed features and is also is for use, thus it was chosen for the simulation, needed for the evaluation of our overlay.

4.1 Simulator

PeerfactSim.KOM is a discrete event-based simulator, that has been developed at the TU Darmstadt. Since then the simulator, that is written in Java, has been improved at the University of Paderborn and the Heinrich Heine University in Duesseldorf. Its main focus is the evaluation of P2P networks. Because of its concept, there is no dependency to specific P2P system and several multi-layer P2P scenarios can be loaded and supported.

4.1.1 Main components

Most of the peer-to-peer simulators have the same basic main components, which have to be present in the structure of the simulator. These basic components will be explained below.

Probably the most used part of every peer-to-peer simulator is the Event Scheduler. It is started before every event occurrence and as its name says, saves a list of future events. These events, which have to happen in the future, are sorted in a list, depending on their time of execution. This list is called Event Queue and is the second component, which is closely linked to the events. These and the other components, needed for the simulation, are part of the Simulation Engine, which is responsible for the management of all components like the host component, the churn generator or the different layers.

The already mentioned layer are in fact network layer, transport layer, overlay layer, application layer and user layer. They are positioned above the Simulation Engine and together with it are building the simulator architecture. At the bottom of the layer structure, the network layer is responsible for the network functionality of the other layers. It simulates the Internet communication between the different peers. The transport layer is

located just over it and its task is to grant the transfer of data between endpoints at the higher levels. The overlay layer is probably the most important layer for us, as it provides the protocols and algorithms needed for the routing and maintenance in the P2P overlays. In the layered structure it is located on the top of the transport layer. Above the overlay layer, there is the application layer, providing the interface for implementing and running applications within the P2P system. At the top of the layer structure, the user layer implements different behavior of the users and thus is giving the possibility to create different simulation models. All the layers are incorporated in the host component. The host component is a representation of a peer. Beside the described layers, there are also the HostProperties as a further part of the host component. The HostProperties can contain the bandwidth, CPU power, RAM etc, depending on the features of the simulation.

4.1.2 Monitoring

The monitoring in the PeerfactSim.KOM is realized through two interfaces - the Monitor and the Analyzer interface. The Monitor has the important task to create a list for every Analyzer and to manage their notifications. The Analyzers area providing information about the network at different time points. This can be the number of hosts, participating on the network, the number of messages they sent and received or other basic information. There some Analyzers, that have already been implemented, however it is also possible to implement own Analyzers, depending on the requirement and the information, needed for evaluation. At the end, the data obtained by the Analyzers, can be saved in files for further analysis or plotted with specific tools.

4.1.3 Configuration

The components, needed for the simulation, are invoked by a configuration file. This is a XML file, which defines the different scenarios. Every configuration file may consist of five main components, which we will present below:
General settings - this part consists of several variables, used later in the configuration file. As they can be defined also in the other parts of the file, the General settings Section is optional.
Simulator core - provides information about the selected seed, used to generate random numbers and the finish time of the simulation.
Components - includes the main components, needed for the current model, like layers, factory class or the default monitor.
HostBuilder - defines the class, used to build the hosts. It sets the groups of peers and their size.
Scenario actions - this part schedules the actions for the simulation.
An example of a configuration file is shown on listing 2. Beside the manual configuration in the configuration file, a GUI can be used to set the parameters of the simulation. As this needs more resources, it won't be discussed in our work.

Listing 2: Example of XML configuration file

```
1  <?xml version='1.0' encoding='utf-8'?>
2  <Configuration>
3
4          <Default>
5            <Variable name="seed" value="3" />
6            <Variable name="finishTime" value="120m" />
7          </Default>
8
9          <SimulatorCore class="org.peerfact.impl.simengine.Simulator"
10           static="getInstance" seed="$seed" finishAt="$finishTime">
11         </SimulatorCore>
12
13         <NetLayer
14           class="org.peerfact.impl.network.modular.ModularNetLayerFactory"
15           downBandwidth="122880" upBandwidth="32768" useRegionGroups="false"
16           useInOrderDelivery="false" preset="Fundamental">
17           <MeasurementDB
18                   class="org.peerfact.impl.network.modular.db.NetMeasurementDB"
19                 file="$gnpDataFile" />
20           <PacketSizing
21                 class="org.peerfact.impl.network.modular.st.packetSizing.IPv4Header" />
22           <Fragmenting
23                 class="org.peerfact.impl.network.modular.st.fragmenting.IPv4Fragmenting" />
24         </NetLayer>
25
26         <TransLayer class="org.peerfact.impl.transport.DefaultTransLayerFactory" />
27
28         <Overlay
29           class="org.peerfact.impl.overlay.dht.edu.geop2p.GeoNodeFactory"/>
30
31         <Monitor class="org.peerfact.impl.common.DefaultMonitor"
32           start="0" stop="$finishTime">
33           <Analyzer class="org.peerfact.impl.overlay.dht.edu.geop2p.analyser.EduLogger" />
34         </Monitor>
35
36         <HostBuilder
37           class="org.peerfact.impl.scenario.DefaultHostBuilder"
38           experimentSize="10000">
39                 <Host groupID="G1"  size="10000">
40                 <NetLayer />
41                 <TransLayer />
42                 <Overlay />
43                 <Properties enableChurn="$churn" />
44           </Host>
45         </HostBuilder>
46
47         <Scenario class="org.peerfact.impl.scenario.CSVScenarioFactory"
48           actionsFile="$actions"
49           componentClass="org.peerfact.impl.overlay.dht.edu.geop2p.GeoNode"/>
50 </Configuration>
```

4.2 Implementation

As already mentioned in Chapter 3, we introduce a motion-aware location-based peer-to-peer overlay in this thesis. The main structure in our overlay is the tree. Therefore we will present the implementation of the areas, as they are used as basis for the tree structure. Furthermore the splitting algorithm from GeoP2P has been adopted and extended. Thus we will introduce the implementation of the splitting method, used in our overlay. Also the implementation of the routing and the movement algorithm will be introduced, as they are crucial for the overall implementation of the overlay.

4.2.1 The GeoArea and the tree structure

The areas, called in our overlay $GeoArea$ and the peers are the main parts of the overlay and also the basis for the tree structure in GeoP2P.Extended. As peers are already implemented in the PeerfactSim.KOM, their implementation would not be further discussed. The peers in GeoP2P.Extended are grouped in GeoAreas. Thus there is class $GeoArea$. Every $GeoArea$ has seven main components, which will be introduced below:

coordinates - As the area is rectangle, there are four coordinates - x_0, x_1, y_0 and y_1.
areaID - The specific ID, showing the place of the area in hierarchy of the tree.
areaContacts - A list of all peers inside the area.
level - The level of the tree, where the area belongs to. This is needed of the update operation.
timeStamp - Another component, that is very important for the update operation, is the timeStamp. It helps the peer to decide if the received information is actual or not.

Beside these five components, there are two other, that are the basis for our tree structure:
parent - As every area, except the root area, has parent area, this one has to be saved in order to have the possibility to move trough the levels of the tree.
children - Again every area, except the leaf areas, has children areas. They are added to the area, after every split operation. Their number is defined by the parameter k, mentioned in 3.2.2

For the maintenance of the tree, several methods are added. The most important of them are:
insertArea - The method, used to add new areas to the tree, e.g. after splitting an area. The method checks if the root area of the tree has any children. If there are not any children, the new area has to be one of the children of the root. If there are children, the method checks if the areaID of the new area starts with the areaID of one of the children. If this is the case, the method checks again if the corresponding child area has any children or not. This is done until area without children is found, such that the areaID of the new area starts with the areaID of the found area.
searchAreaByID - This method is used to obtain information about selected area, e.g. the query to all peers in selected area, mentioned in 3.3. For this purposes similar approach is used, as in the insertArea method. However the goal is to find the area that has the

same ID as the searched ID.

findClosestKnownArea - Another important method is the findClosestKnownArea. This method is crucial for the whole overlay, as it is used for the routing in the join and move operations. Instead of specific area ID, we have the two coordinates of a peer - the x and the y coordinate. The method starts the search again from the root area. If the root area has children, the method checks if the given coordinates of the peer belong inside of one of the children areas. Then at the next step, the method checks again if the child area has children. This is done until a leaf area is found, such that the coordinates of the peer are inside it.

4.2.2 Splitting method

The splitting method is another important method of our overlay. Here we will give a short description of its implementation. Every peer executes the splitting method, when the number of peers in its own area exceeds the threshold. As every peer has own view over the tree and the own area, the splitting has to be done by every peer in the corresponding area.

The first step, when area has to be spit is to compare the x and the y axis. Depending on which of the both axis has to be used for the splitting, its length is divided into k equal parts. The next step is to create k temporally areas and to assign them the coordinates, defined by the division of the longer axis. Furthermore the ID of the divided area is used to define the IDs of the new areas. Every new area becomes also as timeStamp the time of the division. After that the peers, that are in the divided area, are added to the corresponding new area by comparing their coordinate with the coordinates of the new areas. The new areas area also added by every peer to the own view of the global tree. At the end every peer sets the new area, it belongs to, as its new own area.

4.2.3 Routing operation

As already mentioned, the peers has to become the right own area in order to join the network. The right new own area is also very important for the movement around the network, as every peer becomes new area, when it moves to another point of the network. To achieve the right area of selected peer, the routing operation is used. First the peer, that wants to join the network, asks a random peer from the bootstrap list to determine the right area for it. The asked peers executes the findClosestKnownArea method, described in 4.2.1, in order to find the right area for the asking peer. When this is done, a random peer of the found area is selected and the query is resent to it. The new asked peer performs the same operation and if the coordinates of the asking peer area inside its own area, then the asked peer answers to the asking peer with its own area and the own view over the global tree. Otherwise the query is resent to the next right peer.

One specific case is the empty right area. If the found right area has no peers inside it, the

query is not resent and the asked peers answers with the empty area and its own view over the tree, even if it is not inside the right area. The routing operation is a crucial part of both join and move algorithms. The only difference between the join and the move algorithms is that in the move algorithm, the peer has to notify the other peers in its own area about the leaving of the area. Also the old own area has to be added to the list of the old own areas.

4.2.4 GeoConfig

GeoConfig is a specific class, used to configure the parameters of the simulation. It is defining the threshold, used for the splitting of areas. Furthermore the size of the universe, the area update interval, the number of old own areas etc. are defined be the GeoConfig.

4.3 Conclusion

In this Chapter we have presented a short overview of the implementation of our novel location-based overlay. The environment, that has been used to simulate our motion-aware overlay has been introduced in Section 4.1. Further the main component used for the tree structure, the GeoArea, is described in 4.2.1. The method, used for the splitting of the areas is introduced in Section 4.2.2. In 4.2.3 the routing operation, needed for the join and move operations, is presented. As our overlay can be configured for different real-life scenarios, a configuration class was implemented. Description of its implementation can be found in Section 4.2.4.

In Chapter 5 the different scenarios will be presented by different simulation setups. The results of the simulations will be presented in ... and short comparison with the other overlays will be done.

5 Evaluation

Until now, some of the location-based overlays were presented in Chapter 2. In Chapter 3 we presented our novel motion-aware location-based overlay. In this Chapter we will evaluate the quality aspects of our overlay. Furthermore we will compare it with one of the already presented overlays - Globase.KOM [8], as it is very well evaluated by its author. As already mentioned, routing is a crucial part of our overlay, thus we will concentrate on the success of the routing in the overlay. Another important part are the messages, as they are needed for the routing and also the update of the tree structure. As movement of the peers is considered in our work, we will take also a look at the behavior of the overlay in presence of movement of the peers. In the next Section 5.1 the different metrics, used for the evaluation will be introduced. In Section 5.2 the settings of the simulations for the different scenarios will be described. In some of the scenarios, a comparison with Globase.KOM will be made. As we are considering, opposite to Globase.KOM, movement of the peers over the universe, there are few scenarios which are not comparable to the work of Kovacevic and are presented as stand alone evaluation of the characteristics of the overlay in case of motion of the peers. In Section 5.3 the results of the simulation is introduced.

5.1 Metrics

In order to compare our novel overlay with the already existing solutions, quantitative measures of the properties of the overlay will be needed. As Globase.KOM [8] is the better evaluated of the two introduced overlay, it will be used as reference for our evaluation. Thus some of the metrics, described by Kovacevic in [8] will be adopted and evaluated in this thesis. The four main metrics, that will be used, will be described below.

5.1.1 Success of lookups

As routing is crucial for our overlay, the total number of lookups and the number of successful and failed lookups will be compared. The results will be obtained by the analyzer, already described in 4.1.2.

5.1.2 Sent and received messages

Similar to the number of the lookups, we have to obtain the number of sent and received messages. The resulting number of failed messages is also important for the evaluation of the overlay. The same analyzer will be used for this and also for the next metrics.

5.1.3 Number of hops

The number of the hops for a query shows us how many peer has to be approximately contacted in order to receive the right area. The results of this metric are hardly depending of the predefined threshold. Theoretically bigger threshold has to cause smaller number of hops.

5.1.4 Depth of the tree

This metric is again tightly connected with the predefined threshold and thus it is crucial for the speed of the routing algorithm. Similar to the number of hops, smaller threshold will result in deeper tree structure.

5.2 Scenarios

As already mentioned, the peer-to-peer simulator PeerfactSim.KOM will be used in our simulations. This will help us to provide results, that are similar to the results that can be obtained from real world situation. Using the build-in random generator of the simulator, we will run every simulation with 10 different seeds and thus our graphics will always present the average of the runs of the 10 seeds.

First we will compare some of the characteristics of our overlay with Globase.KOM. For this purposes configuration of the overlay attributes, similar to the presented by the author of Globase.KOM, will be used. As movement of the peers is not considered in Globase.KOM, comparison of the characteristics of the overlay in presence of moving peers will not be possible. Thus a stand-alone evaluation of these characteristics will be done. This will be divided into two parts. First we will take a look at the results in a more realistic situation, where peers are joining the network and moving at the same time. Furthermore we will also evaluate the behavior of the network in case the already joined peers are just moving, without joining of new peers.

In order to evaluate the characteristics of the network in presence of moving peers, churn and packet loss will not be considered in our evaluation, as they will distort the evaluation of the behavior of the overlay. In Section 5.2.1 the results of the evaluation of a static network and the comparison with Globase.KOM is presented. Further in Section 5.2.2 and 5.2.3 movement of the peers is considered, respectively together with the join of the peers or as a stand alone event.

5.2.1 Static peers

In the first group of scenarios, movement of the peers in the network will not be considered and thus a good comparison with Globase.KOM will be possible. As our overlay has

some parameters, that can be predefined, we expect that these will play main role in the behavior of the overlay. Some of them are similar to the other solutions, other are missing in them. As the peers in our overlay have equal roles and we don't have techniques, similar to the interconnections of Globase.KOM, which are described in Section 2.4.1, we will not consider the result of Globase.KOM, when interconnections are used. On the other side, the impact of the size of the threshold parameter, which defines when an area has to be split, can be evaluated and compared to the work of Kovacevic, which is also done. For these purposes we evaluate the average number of hops per query for different size of the threshold, compared to the whole number of peers. The different scenario settings are shown on Table 1. We have 4 settings, that will stay constant in this scenario, as they are not also considered in the evaluation of Globase.KOM - the simulation time, the time between updates of information about the tree and movement and churn. On the other side the number of the peers will be 1000 or 10 000 and the threshold will vary between 10 and 100, with step of 10 between the values. This will help us to evaluate the effect of the different form of the three on the results.

Table 1: Simulator settings for evaluation of the static scenario

Settings	**Value**
Number of peers	1000 (10 000)
Time	60 minutes
Threshold	10,20,30,...,100
Updates	every 60 seconds
Movement	None
Churn	None

5.2.2 Real life movement

The second group of scenarios is called real life movement scenarios, because some of the peers will start their movement as other peers are still joining the network. This situation is the nearest to the real world and will give us a good overview of the behavior of the overlay in real life situation. In this part comparison of some characteristics of the overlay against Globase.KOM will be done again, however they will be not so representative for the performance of our overlay, as Globase.KOM is not considering movement of the peers. Similar to the first group of simulations we will set the number of peers to 1000 and 10 000 and again the threshold will be between 10 and 100 with the step of 10 between the different values. This time movement of the peers will be turned on and every peer will be allowed to move. The number of peers allowed to move can be set with a parameter in the config file of our overlay, but the different parameters and their role in the performance will be evaluated in Section 5.2.4. In this group of scenarios the movement will be possible after simulation minute 15, because we will need some peers to join the network and form the areas first. Again on Table 2 the different settings for the simulator are shown.

Table 2: Simulator settings for evaluation of the real life scenario

Settings	**Value**
Number of peers	1000 (10 000)
Time	60 minutes
Threshold	10,20,30,...,100
Updates	every 60 seconds
Movement	From minute 15
Churn	None

5.2.3 Stand alone movement

This group of scenarios is probably the most interesting, as it will give us an overview of the performance of our movement algorithm. When evaluating the results of these scenarios, no comparison against Globase.KOM will be done, as they will be used for the evaluation of the behavior of motion-aware location-based peer-to-peer overlays and in the work of Kovacevic movement of the peers is not considered. Also there will be some other differences in the settings. First the simulation time will be longer - 180 minutes in this case. In the first 70 minutes the peers will join the network, but will stay static. After that we are giving 20 minutes simulation time for stabilization of the network. In the last 90 minutes the peers will start to move over the universe. Here we will have also different changing parameter - the number of peers, that are allowed to move over the universe. They will be 10% and 100% of the whole number of peers in the network. The size of the network will be 10 000 and the threshold will be set to 50 and to 100. Churn will not be considered again. The simulation settings are shown on 3.

Table 3: Simulator settings for evaluation of the stand alone movement scenario

Settings	**Value**
Number of peers	10 000
Time	180 minutes
Threshold	50 (100)
Updates	every 60 seconds
Movement	From minute 90
Churn	None

5.2.4 Changing parameters

This is a small group of simulations, where we investigate the impact of the difference configuration parameters on the behavior of the overlay. For this purposes a constant number of 1000 peers will be used. The simulation time will be 60 minutes and also movement will not be considered. However the different parameters, like update level and update interval will be changed in the different simulations.

5.2.5 Churn

Even if churn is part of our future work, a short test of our overlay in presence of churn will be done. We expect increased number of failed lookups, especially if peers move around the network. For the evaluation of the churn scenario a group of 1000 peers will be used and also simulation time will be set to 120 minutes. We will have two cases here - churn in static overlay and churn in presence of motion of the peers. In order to differentiate these both cases, we will use similar conditions as in the stand alone movement scenario. In the first 60 minutes the peers will join the network. From minute 65 until the end all the peer will be allowed to move.

5.3 Results

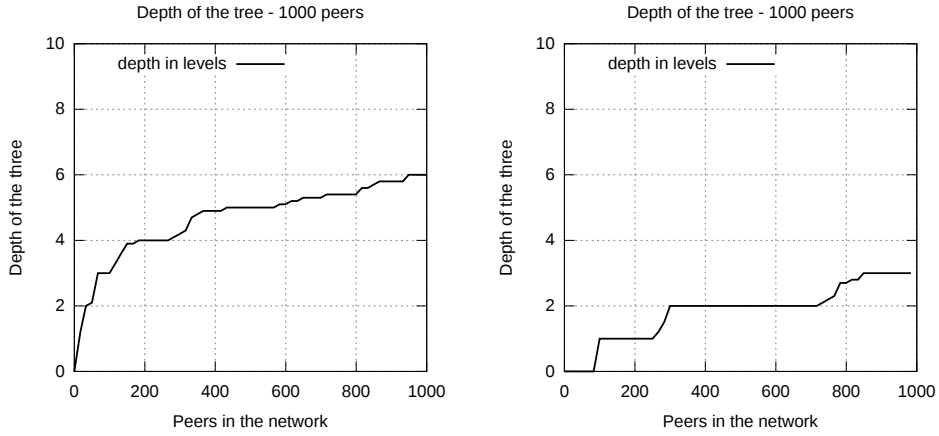
In the next sections, we evaluate the results of our simulations. In Section 5.3.1 we do not consider motion of the peers and thus a comparison with Globase.KOM is possible and also done. Probably the most interesting characteristic - the number of hops needed for a query are evaluated. Further in Section 5.3.2 and Section 5.3.3 the behavior of the overlay in presence of motion of the peers is evaluated. In Section 5.3.4 the impact of the different configuration parameters on the overlay is investigated. In the last Section 5.3.5 short evaluation of the overlay in presence of churn is done.

5.3.1 Static peers

In this Section we show the results of the simulations of the first scenario. First the depth of our tree is shown, when the threshold is 1% of the whole number of peers 18a and when it is 10% 18b. These are the two extreme values of our simulations and we expect that they will result in different depth of the tree, which can impact the other characteristics of the overlay. The results are expected, especially when we consider that we have a constant breadth of the tree, depending on the predefined number of subareas of every area, which is 3 in this case.

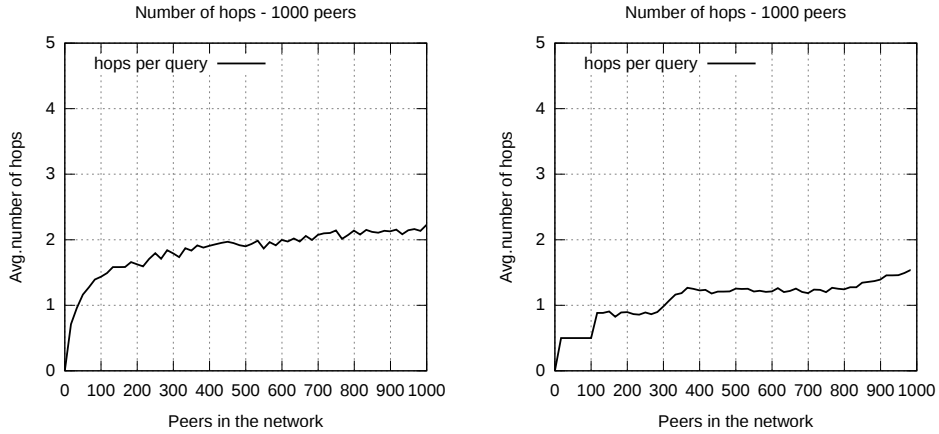
As we do not have interconnections and superpeers and also there is constant breadth of the subtrees, the number of hops needed for a query depends on the depth of the tree as we can see on Figure 19a and Figure 19b. Especially when the threshold size is 10% of the whole number of the peers and we do not have so many splittings of areas, the dependency between the depth of the tree and the average number of hops is easy to see. Again - the threshold is a predefined parameter, which describe when an area has to be split in subareas. When the number of peers in an area exceeds the predefined threshold, the area is split into subareas.

Even if the results of the simulation of our overlay are more than satisfying and it is giving us good performance, especially considering the number of hops needed per query, it is much more interesting to compare our overlay with the existing solutions. On Figure 20 we can see the average number of hops per query with changing threshold size. We ran



(a) Threshold = 1 percent (b) Threshold = 10 percent

Figure 18: Depth of the tree - static scenario



(a) Threshold = 1 percent (b) Threshold = 10 percent

Figure 19: Average number of hops per query - static scenario

the simulation with 10 different thresholds - with 10 and 100 and as we have 1000 peers, these are respectively 1% and 10% of the whole number of peers. Similar simulation has been done by Kovacevic in [8] and compared to the Globase.KOM our overlay needs significantly less hops for a query. Second, and this is more interesting, the number of the needed queries constantly falls with rising size of the threshold and thus we do not have the peak, that can be seen in Globase.KOM, when the threshold's size equals 2% of the whole number of peers as shown on Figure 20.

Another interesting point of comparison is the load balance ratio. This is defined by the author of Globase.KOM as a division of the load of the most loaded peer by the load of the median loaded peer. The number of received messages is used as metric for load. As all the peers in our overlay have the same role, we expect uniformly distribution of the load. On Figure 21 we can see this comparison between GeoP2P.Extended and Globase.KOM.

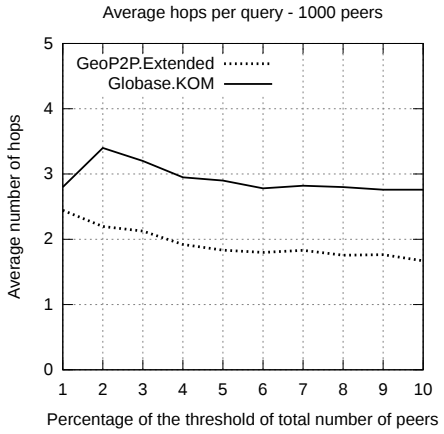


Figure 20: Average hops per query

When the threshold represents between 7 and 10% of the whole number of peers, there are almost no differences and the load balance ratio is about 2, which is speaking for very good balance of the load between the peers.

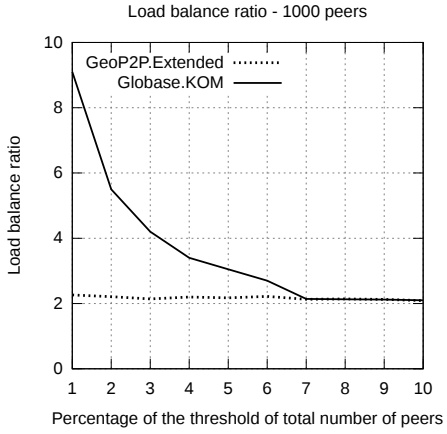
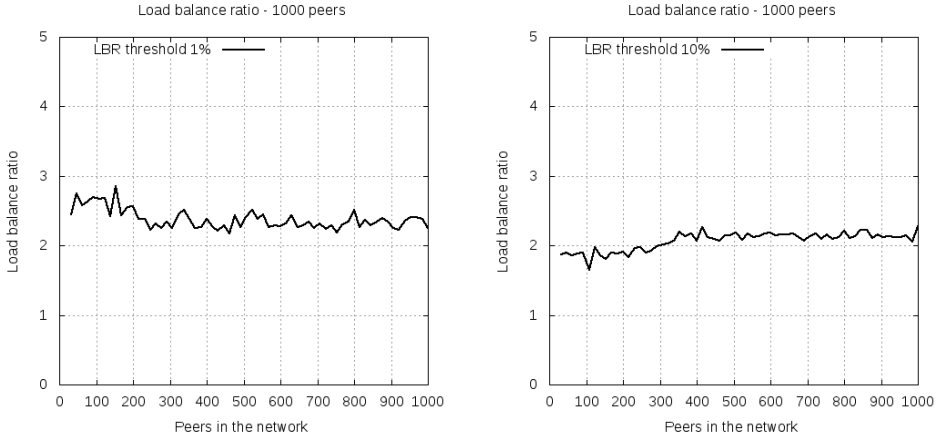


Figure 21: Load balance ratio - static scenario

However if the threshold represents less than 7% of the number of peers in the network, the load balance ratio of Globase.KOM rises with decreasing size of the threshold and is achieving values of about 10, when the size of the threshold is 1% of the whole number of peers. The load balance ratio of GeoP2P.Extended stays constant with values about 2, even if the size of the threshold is only 1% of the network size. One reason for these differences is the different breadth of the tree and at this point the constant number of subareas, which every area is split in, seems to be a big advantage of GeoP2P.Extended.

As the load balance ratio of our overlay seems to be stable and predictable with changing

size of the threshold, we decided to track the load balance ratio during changing number of peers. On Figure 22 we can see the load balance ratio when the number of peers rises from 0 to 1000 in the 2 extreme threshold sizes - 1% of the whole size and 10% of the whole size. As shown on Figure 22a, the load balance ratio stays again almost constant, when new peers join the network. Again the values between 2 and 3 are speaking for a good balance of the load between the peers. The same conclusions can be done on Figure 22b, where the threshold has the other extreme size - 10% of the whole network size. Again the load balance ratio stays constant and is between 1.8 and 2.3.



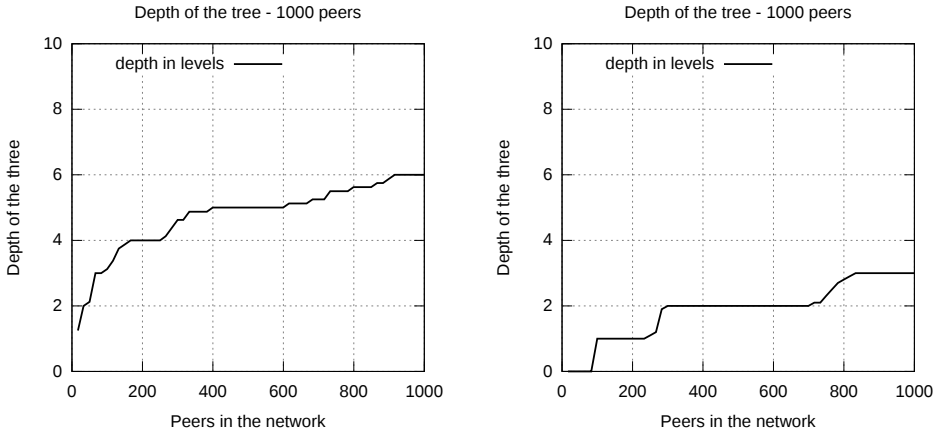
(a) Threshold = 1 percent (b) Threshold = 10 percent

Figure 22: LBR - changing threshold, static scenario

5.3.2 Real life movement

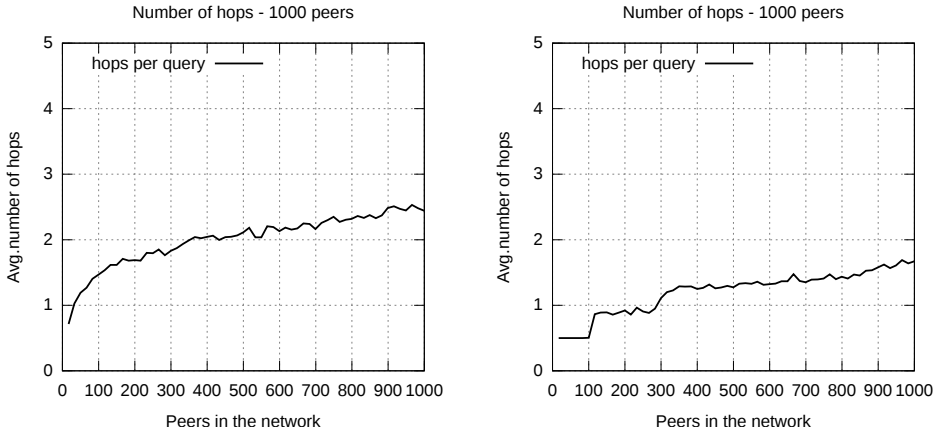
In this Section we consider the movement of the peers in nearby real life situation, where peers join the network and move around at the same time. Similar to the previous Section the simulation time is 60 minutes. On Figure 23a we can see the depth of the tree with rising number of peers in the network, when the threshold is equal to 1% of the network size. Compared with the results of the previous section, we can see that the movement of the peers does not impact the depth of the tree. We get the same result also for a threshold with size 10% of the whole number of peers as shown on Figure 23b.

The movement of the peers has a little bit more impact on the average number of hops, needed per query. In Figure 24a we can see that the average number of hops, if the threshold size represents 1% of the network, has raised from about 2 when the peers are static to 2.5 when the peers are moving around the network. Even if there is a little difference, this is still not big impact on the performance of the overlay. At the other side, with rising threshold size, e.g. threshold = 10% like in Figure 24b, the difference between the static and motion-aware overlay is getting smaller and is not playing such a role anymore. We are still achieving a good performance for the queries, as peers move around the network.



(a) Threshold = 1 percent (b) Threshold = 10 percent

Figure 23: Depth of the tree - real life scenario



(a) Threshold = 1 percent (b) Threshold = 10 percent

Figure 24: Average number of hops per query - real life scenario

Even if the author of Globase.KOM does not consider movement of the peers, we decided to evaluate the average number of hops of our overlay under changing threshold size and to compare it with the evaluation done by Kovacevic in [8]. The results are shown in Figure 25. As we do not have any significant differences between the values of our static and motion-aware scenarios, it is not surprising that also the average number of hops per query, considering the changing threshold size, has not changed significantly. Again the number of hops per query falls constantly, with no peaks and also values under 2 at most of the time.

Probably the most interesting point, together with the number of hops, is the load balance ratio. Surprisingly, even if the peers are moving around the network, the load balance ratio of our overlay stays constant with values around 2, as we can see in Figure 26. This was not expected, because when peers move, they start queries for every move and

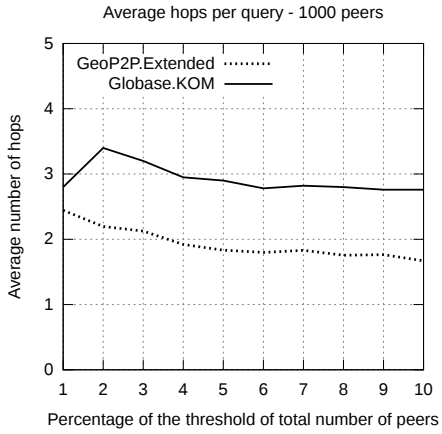


Figure 25: Average hops per query - real life scenario

respectively send messages to the peers in the new areas and thus we expected that with increasing number of moves, the load balance ratio would get bigger values, meaning worse load balance ratio.

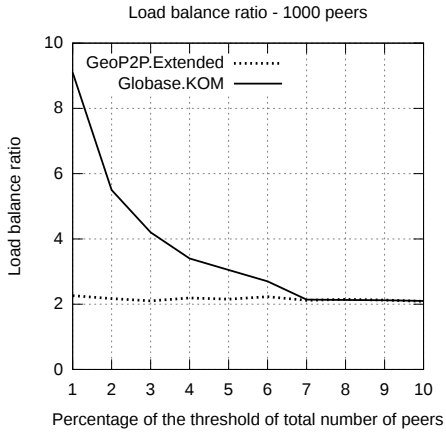
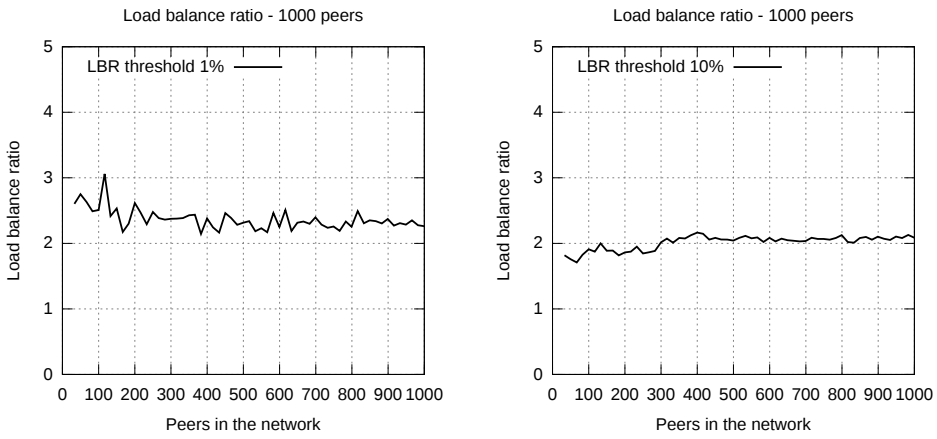


Figure 26: Load balance ratio - real life scenario

Again, when we have threshold size between 1% and 7% of the network, the load is much more evenly distributed, as this is the case in Globase.KOM. With rising threshold size, the load balance ratio of the overlays is getting almost equal, similar to the results, that we got in Section 5.3.1. Also if we compare the load balance ratio, assuming rising number of peers in the static and the motion-aware scenario, we can see that there are only small and not significant differences. The results for the two extreme threshold - 1% and 10% are shown respectively in Figure 27a and Figure 27b.



(a) Threshold = 1 percent (b) Threshold = 10 percent

Figure 27: LBR - changing threshold, real life scenario

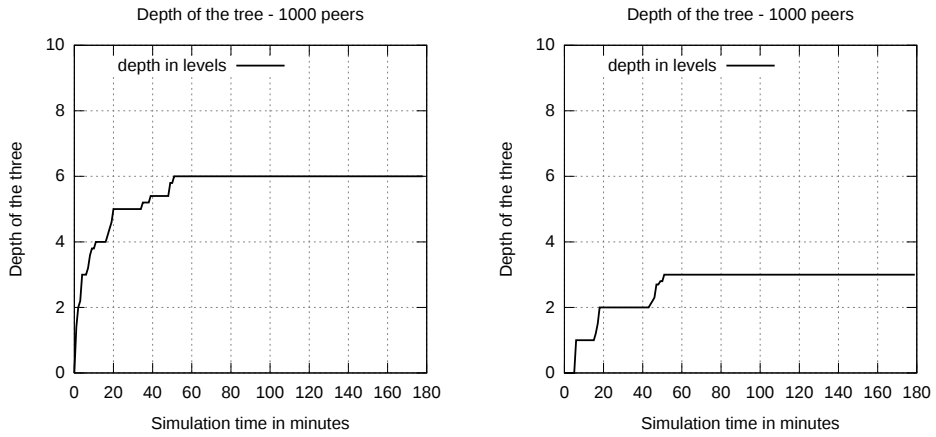
In this Section we have discussed the behavior of our overlay under real life conditions. However one have to notice that the definition "real life conditions" is a little bit overdimensioned, as there is almost no real life situation, where all the peers are moving around. We choose this heavy load scenario to evaluate the behavior of the overlay under full load, when peers are moving and joining at the same time. In real life situation, probably every second or every third peer will move around the network. These situations are discussed in Section 5.3.4, where we evaluate the effect of the changing values of the different parameters in our GeoConfig class.

5.3.3 Stand alone movement

In order to differentiate the result of the evaluation, done in Section 5.3.2, we have simulated a separate scenario, where the movement of the peers starts after all the peers have joined the network. In this scenario group the simulation time has been increased to 180 minutes. In the first 60 minutes the peers are just joining the network, then they pause 10 minutes in order to update their information about the network. Using a random generator, we start the movement of the peers from minute 70. Again every peer is allowed to move, and the movement start point depends on the random generator. Opposite to the previous section, we present the results per time and not per peer, as we have all 1000 peer at minute 60 in the network, but the actual evaluation of the movement starts from minute 70.

First we want to consider the influence of the movement on the tree structure. As the breadth of the subtrees is predefined by a parameter, the depth value is interesting for us. In Figure 28a we can see the depth of the tree, when the threshold size equals 1 % of the network size. In Figure 28b the results from the same simulation for threshold = 10% are shown. The most interesting thing is that the depth of the tree on both figures rises with rising number of peers until minute 60 of the simulation, but does not change from

minute 70 until the end of the simulation - the period when the peers are moving around the network. This is one of the main advantages of our splitting method. As areas are split in equal subareas, there are also areas that does not have so many peers until minute 60. When peers move into these areas later, after minute 70, there is still enough place in the areas and they do not have to be spit and so the depth of the tree does not change. Also the areas area not dynamically merged and this seems to be a disadvantage of our solution in the first moment. However, if we consider our scenario with the animals, we have to have in mind, that animals are periodically migrating between same points, e.g. south and north half of the earth, and thus merging the areas and splitting them again, when the peers are back will cause much more load, as saving the information about the split areas into the routing table.

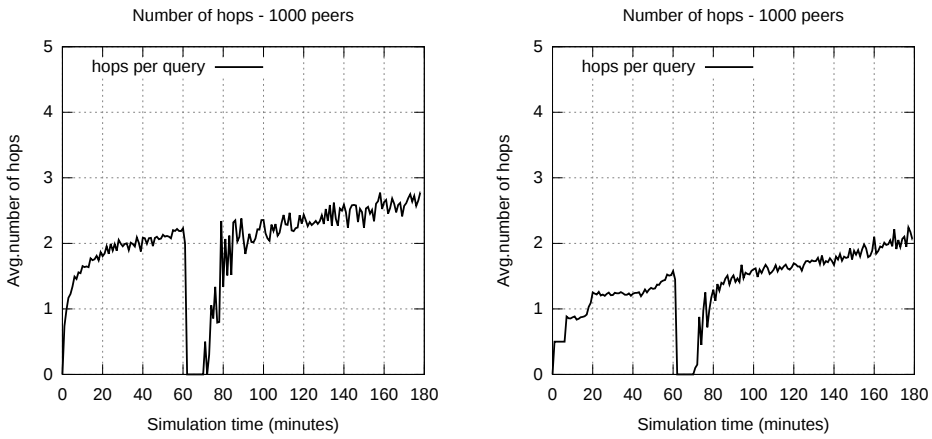


(a) Threshold = 1 percent (b) Threshold = 10 percent

Figure 28: Depth of the tree - stand alone movement

The next evaluation point is again the average number of hops needed for a query. As we can see in Figure 29 the constant breadth and depth of the tree results in an almost constant average number of hops per query. This is very easy to see especially in Figure 29a, where the threshold size is representing 1% of the network size. First, until minute 60 we have similar results as in the previous scenarios, where the peer where static or in real life situation. Then, because of the pause between minute 60 and 70, there are no queries in this period. From minute 70 the average number of hops rise again with rising number of moving peers, but is not getting bigger than 2.1-2.2 - values that we had also in the static scenarios.

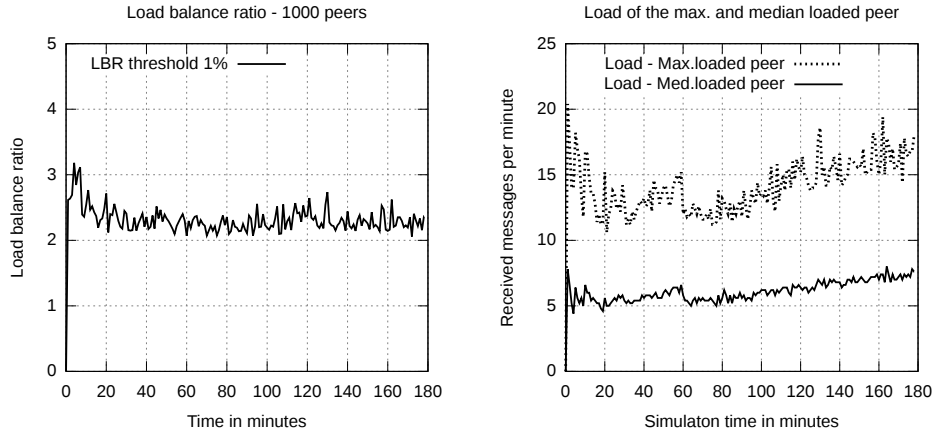
The last metric that is important for us, when we evaluate the movement of the peers, is the load balance ratio. In Figure 30a we can see that movement of the peers has no impact on the load balance ratio, when the threshold is 1% of the whole network size. As peers update their routing tables and still send messages also in the pause between minute 60 and 70, it is not easy to see if there is any influence of the movement on the load of the peers. Thus we decided to show also the load of the most loaded and median loaded peer in Figure 30b. As we can see the load for the most loaded and for the median loaded peer falls between minute 60 and 70, but because of the update operation there is still load for every peer. But as the relation between the load of the most loaded and



(a) Threshold = 1 percent (b) Threshold = 10 percent

Figure 29: Average number of hops per query - stand alone movement

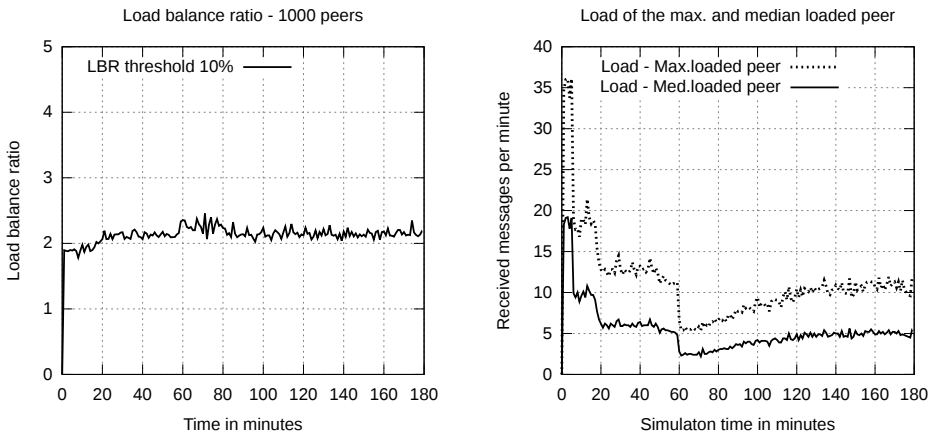
median loaded peer stays constant, the load balance ratio stays also almost constant with values similar to these, that we already know from the static and real life scenarios.



(a) (b)

Figure 30: Load balance ratio - stand alone movement 1%

The same simulation has been done for a threshold size - 10% of the network size, giving us similar results.



(a) (b)

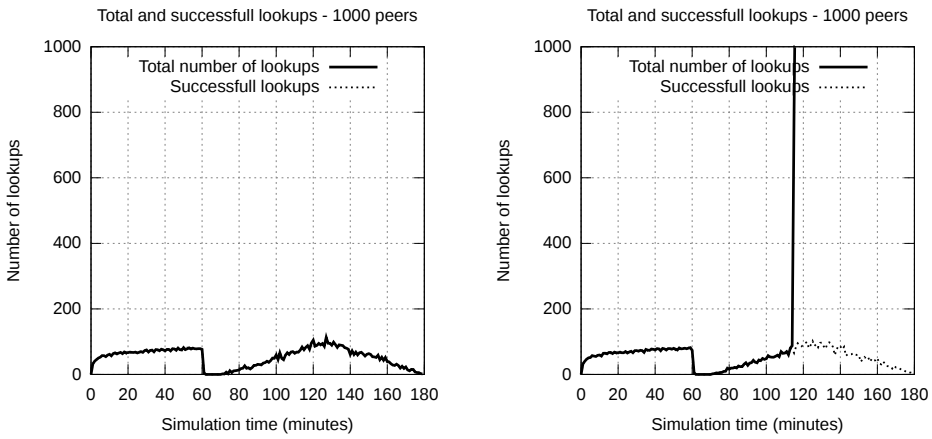
Figure 31: Load balance ratio - stand alone movement 10%

5.3.4 Changing parameters

As already mentioned, there are some parameters for the simulation, which can influence the behavior of the overlay. To evaluate the impact of the parameters in our GeoConfig class, we use network with size 1000 peers. As threshold size and its impact on the performance was already discussed in the previous section, we do not consider changing threshold size and it is set to 50, meaning 5% of the whole network size. Further we use the time setting from the Section 5.3.3 in order to differentiate the results in static and motion-aware state of the network.

First we take a look at the parameter $updateInterval$, which defines how often a peer updates its routing table. When peers do not move, there is no impact by this parameter. However this parameter is very important if the peers are allowed to move. If the peers do not update their routing tables often enough, they do not have actual information about the network, they can not provide the right contact peers in case of arrived query. This causes increased number of lookups and not functional network. On Figure 32a we can see, that if the peers update their routing table every minute, the network stays stable, even if all the peers are allowed to move and under heavy load, especially between minutes 110 and 140. When we set the $updateInterval$ to 10 minutes, there are no problems as long as peers do not move, as this is the case until minute 65. If a high number of peers start to move and they do not update their routing tables, the number of lookups raises very fast from minute 115 and causes large load of about 30 000 lookups per minute and almost complete failure of the network, as we can see in Figure 32b. Even if the network is able to get stable also in presence of such big load, it is easy to see that the $updateInterval$ is crucial for the stability of the network, if peers are moving.

The next parameter - $notifyLevel$, describes how many levels above the own level has to be informed about the own movement to another point of the universe. As we can see on Figure 33, this parameter does not cause any big impact on the behavior of the overlay.



(a) Update every minute (b) Update every 10 minutes

Figure 32: Impact of the parameter $updateInterval$

It only cause small differences on the average number of hops, needed when peers are moving after minute 70 of the simulation.

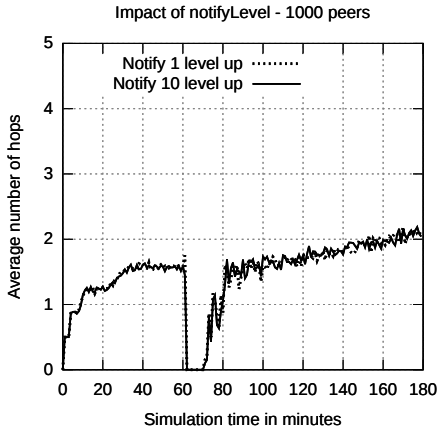


Figure 33: Impact of the parameter $notifyLevel$

As expected the parameter, that is causing the biggest impact on the overlay together with the updateInterval, is the parameter that defines the moving behavior. This is the $moveCounter$ parameter, describing how many moves a peer is allowed to do in the simulation. On Figure 34 we can see, that the number of times peer is allowed to move almost equals the increase of the number of lookups.If peers are allowed to move up to 3 times, during the simulation, they cause about 3 times more lookups, than peers that are allowed to move only once.

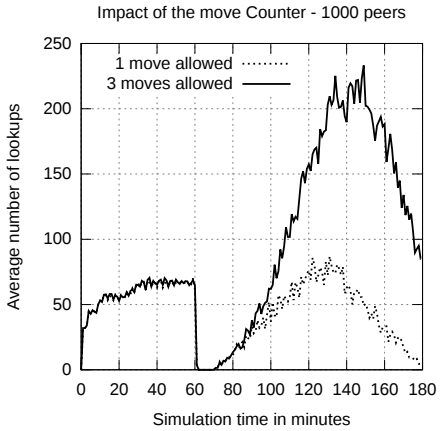
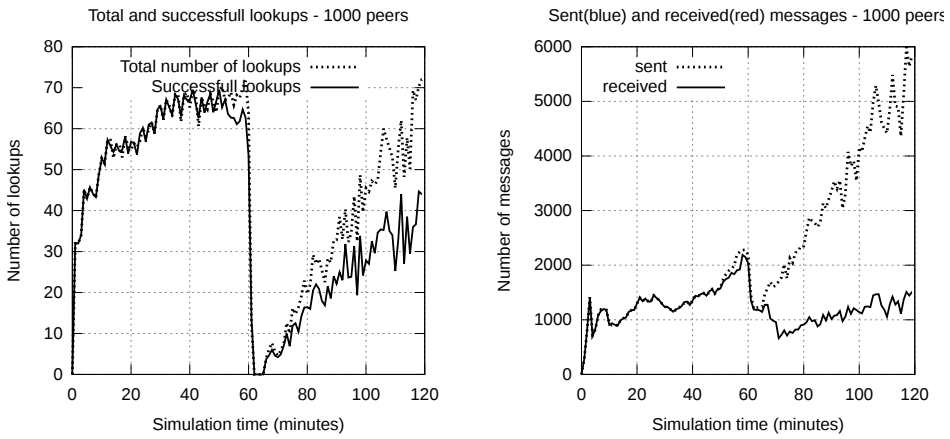


Figure 34: Impact of the parameter $moveCounter$

5.3.5 Churn

As already mentioned, considering churn is part of our future work and thus the results of the simulation are reasonable. First churn does not significantly impact characteristics of the overlay like the depth of the tree or average number of hops. As we can see in Figure 35a, there is almost no impact also on the ratio of the number of successful lookups to the total number of lookups if the peers do not move, as this is the case until minute 60 of the simulation. However when the peers start to move from minute 65, the number of successful lookups decreases and represents about 60% of the total number of lookups at the end of the simulation. Considering the number of sent and received messages, the difference is getting even bigger and at the end of the simulation only about 30% of the sent messages arrives at the receivers as shown in Figure 35b. The difference between the ratio of successful and total lookups and the ratio between send and received messages is easy to explain. As every peer makes up to 3 attempts to send a message for each lookup, it is normal to have up to 3 times more messages that are not received, as lookups that are not successful. Much more interesting to find out the reason for the missing success of about 30% of the lookups. As peers move around, the peers in every area are continuously changing and thus finding the right contact is not an easy task. With the help of our update algorithm and also the oldOwnAreas structure, described in Section 3.4.2, this problem is solved if all the peers are online. If the peers get offline, finding the right contact is not always possible at the first attempt, as peers get information about offline contact only when they ask him or if they make an update. However if the offline peer goes offline between the update operation, the information about its offline status is not provided to all the peers in the network. However if an offline peer can not be reached, the next best known peer is chosen, helping to find the right area even in presence of churn.



(a) (b)

Figure 35: Impact of churn in GeoP2P.Extended

5.4 Conclusion

In this Chapter we have investigated the behavior of GeoP2P.Extended under different conditions and also compared it with Globase.KOM. The main metrics used for the evaluation are presented in Section 5.1. Our simulation scenarios have been presented in Section 5.2. They are divided in 3 major groups and also 2 smaller groups of scenarios. First in Section 5.2.1 we compare our overlay with Globase.KOM without considering movement of the peers. In Section 5.2.2 the movement of the peers in real life situation is considered. In Section 5.2.3 we divide the states of the overlay in static and motion-aware in order to evaluate the movement of the peers, without considering joining of the network. Further in Section 5.2.4 the impact of the different parameters in our GeoConfig class is evaluated and at the end the behavior of the overlay in case of churn is evaluated in Section 5.2.5.

The results of our simulation has been discussed in Section 5.3. In Section 5.3.1 we can see that our overlay performs very well if peer do not move around the network. As the goal of this Thesis was to investigate the behavior of peer-to-peer location-based overlays in case of motion of the peers, we have evaluated the motion in Section 5.3.2 and 5.3.3. Again the results has shown that a peer-to-peer network can stay stable, even if participants move around, if there is no churn. In Section 5.3.4 that the parameters, which describe the moving behavior of the peers, play a significant role in the behavior of the whole overlay.

As churn was not part of our research, it was expected that we will have a big amount of not successful lookups and lost messages and thus the results of the simulation of our overlay in case of churn in Section 5.3.5 are not surprising. Churn can be a big problem for motion-aware peer-to-peer networks and thus a good stabilization algorithm is needed. This will be part of our future work.

6 Conclusion

In this thesis we try to answer the question about the behavior of location-based peer-to-peer network, when peers are moving. In order to answer this question, we first compare existing location-based overlays. On the one side is the heterogeneous solution, presented by the author of Globase.KOM in [8], on the other is GeoP2P [9], where all the peers have equal roles. Unfortunately none of these solution considers the movement of the peers. This is the reason for us to develop a new overlay - GeoP2P.Extended, using the static GeoP2P as basis for our implementation. An event-based peer-to-peer simulator was used for the implementation of our solution and at the end a comparison with the existing overlays has been done in different simulation scenarios.

We show that movement of the peers around the network impacts the number of lookups and also the number of messages, that are sent and received by the peers, causing more load for the network. Furthermore update and stabilization algorithms are needed in order to assure the completeness and correctness of the queries and also of the overlay structure. For these purposes periodical updates and also a simple structure, saving the information about the areas, a peer has been into, is needed. Also we show, that the number of moving peers plays significant role in the behavior and the stability of the overlay.

At the end of the work our GeoP2P.Extended overlay is evaluated and compared with Globase.KOM [8]. For these purposes we have five different groups of scenarios. First, we evaluate in its static state and compare it with Globase.KOM. Further the behavior of the overlay in presence of movement is evaluated. Here two situations are considered - real life scenario, where peer are joining and moving at the same time and stand alone scenario, where movement is separated from the join process in order to evaluate only the movement. Also the different configuration parameters and the behavior in presence of churn are evaluated in the last groups of scenarios.

Future work
In this thesis we focus on the behavior of location-based peer-to-peer overlays in presence of moving peers. We consider the success of the queries and also the stability of the tree structure, which stays behind our overlay. It would be interesting to investigate the behavior of such overlays in presence of churn, which will represent more realistic situations. Thus an stabilization algorithm, considering the movement of the peers will be needed. With the help of this, it will be possible to obtain more actual information about the network and to decrease the number of lost messages and also queries, which are not successful. As current solutions do not consider movement of the peers, we need to find out which requirements have to be met in order to implement such stabilization algorithm.

References

[1] A. J. W. Andrew C. Markham, "Ecolocate: A heterogeneous wireless network system for wildlife tracking,"

[2] CSAIL, "Binary space partitions, or, doom in 20 minutes." http://groups.csail.mit.edu/graphics/classes/6.838/S98/meetings/m13/bsp.html. Accessed: 2015-11-17.

[3] OpenDSA, "Opendsa hypertextbook project." http://algoviz.org/OpenDSA/Books/Everything/html/KDtree.html/. Accessed: 2015-11-13.

[4] C. Dillabaugh, "Efficient batch query processing on r-trees using floating buffers." http://cglab.ca/~cdillaba/comp5409_project/R_Trees.html/. Accessed: 2015-11-11.

[5] M. Chakravarty, "The use of context in pattern recognition." http://www.bic.mni.mcgill.ca/~mallar/CS-644B/Home.html/. Accessed: 2015-10-11.

[6] P. K. Gabriel Ghinita and S. Skiadopoulos, "Mobihide: A mobilea peer-to-peer system for anonymous location-based queries,"

[7] Q. H. Vu, "Sgr-tree: a skip graph based r-tree for multi-dimensional data indexing in peer-to-peer systems,"

[8] A. Kovacevic, *PEER - TO - PEER LOCATION - BASED SEARCH: ENGINEERING A NOVEL PEER - TO - PEER OVERLAY NETWORK*. PhD thesis, Technischen Universität Darmstadt, 2009.

[9] S. Asaduzzaman and G. v. Bochmann, "Geop2p: An adaptive peer-to-peer overlay for efficient search and update of spatial information,"

[10] "IUCN Red List table 3a." http://cmsdocs.s3.amazonaws.com/summarystats/2015-4_Summary_Stats_Page_Documents/2015_4_RL_Stats_Table_3a.pdf. Accessed: 2015-12-13.

[11] M. Wikelski, R. W. Kays, N. J. Kasdin, K. Thorup, J. A. Smith, and G. W. Swenson, "Going wild: what a global small-animal tracking system could do for experimental biologists,"

[12] O. Heckmann, "Qualitätsmerkmale von peer-to-peer-systemen. technical report kom-tr-2006-03," tech. rep., Technischen Universität Darmstadt, 2006.

[13] B. F. Naylor, "A tutorial on binary space partitioning trees,"

[14] M. Knoll and T. Weis, "Optimizing locality for self-organizing context-based systems." http://www.win.tue.nl/~hermanh/doku.php?id=recursive_tilings_and_space-filling_curves. Accessed: 2015-12-17.

[15] C. Gotsman and M. Lindenbaum, "On the metric properties of discrete space-filling curves." IEEE Transactions on Image Processing, 1996.

[16] P. K. J.K.Lawder, "Using space filling curves for multidimensional indexing,"

[17] D. K. M. K. H. B. Ion Stoica, Robert Morris, "Chord: A scalable peer-to-peer lookup service for internet applications"

[18] P. D. Antony Rowstron, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems"

[19] J. S. S. C. R. A. D. J. J. D. K. Ben Y. Zhao, Ling Huang, "Tapestry: A resilient global-scale overlay for service deployment,"

[20] S. L. Changxi Zheng, Guobin Shen and S. Shenker, "Distributed segment tree: Support of range query and cover query over dht,"

[21] H. Haverkort, "Recursive tilings and space-filling curves." Self-Organizing Systems. Published: 2006.

[22] L. R. Filipe Araujo, "Geopeer: A location-aware peer-to-peer system"

[23] M. H. R. K. Sylvia Ratnasamy, Paul Francis and S. Shenker, "A scalable content-addressable network"

[24] H. G.-M. Prasanna Ganesan, Beverly Yang, "One torus to rule them all: Multi-dimensional queries in p2p systems"

[25] B. Waresiak and P. Skrzynski, "Using quad tree as data storage for a terrain representation and a core for a path finding algorithm"

[26] J. Aspnes and G. Shah, "Skip graphs"

[27] M. K. Anirban Mondal, Yilifu, "P2pr-tree: An r-tree-based spatial index for peer-to-peer environments"

[28] R. Y. Chi Zhang, Arvind Krishnamurthy, "Brushwood: Distributed trees in peer-to-peer systems"

[29] R. Y. Chi Zhang, Arvind Krishnamurthy, "Skipindex: Toward a scalable peer-to-peer index service for high dimensional data"

[30] D. A. Tran and T. Nguyen, "Hierarchical multidimensional search in peer-to-peer networks"

[31] R. C. Doyle, "Distributed bootstrapping of peer-to-peer networks"

[32] F. Murtagh and P. Contreras, "Methods of hierarchical clustering" tech. rep., Department of Computer Science, Royal Holloway, University of London, 2011.

[33] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric"

[34] J. L. R. M. Thomer M. Gil, Frans Kaashoek and J. Stribling, "P2psim a simulator for peer-to-peer protocols." `https://pdos.csail.mit.edu/archive/p2psim/index.html`. Accessed: 2015-12-12.

[35] G. P. J. S. V. Mark Jelasity, Alberto Montresor, "Peersim: A peer-to-peer simulator." http://peersim.sourceforge.net/. Accessed: 2015-11-12.

[36] M. Feldotto and K. Graffi, "Comparative evaluation of peer-to-peer systems using Peerfactsim.KOM"