Geographical Location of Internet Hosts using a Multi-Agent System

Øystein Espelid Thorvaldsen

Master of Science in Computer Science
Submission date: November 2006
Supervisor: Svein Johan Knapskog, ITEM
Co-supervisor: André Aarnes, Q2S

Norwegian University of Science and Technology
Department of Telematics
Problem Description

The student will conduct an experimental study on the use of multi-agent systems in Internet Investigations. Based on a prototype developed as part of a previous project, the student will develop and test a prototype for geolocation of IP addresses using multi-agent technology. Through practical experiments using the Uninett research infrastructure, the student will evaluate the performance of the prototype and compare the results to other existing geolocation methods. The student is encouraged to propose novel methods or improvements based on the experiments performed. The project is given in cooperation with the High Tech Crime Division at the National Criminal Investigation Service (Kripos).

Supervisor: Svein Johan Knapskog, ITEM
Geographical Location of Internet Hosts using a Multi-Agent System

Øystein E. Thorvaldsen
Department of Computer and Information Science
Norwegian University of Science and Technology

November 15 2006
Abstract

This thesis focuses on a part of Internet forensics concerned with determining the geographic location of Internet hosts, also known as geolocation. Several techniques to geolocation exist. A classification of these techniques, and a comparative analysis of their properties is conducted. Based on this analysis several novel improvements to current techniques are suggested.

As part of an earlier designed Multi-Agent Framework for Internet Forensics (MAFIF), an application implementing two active-measurement geolocation techniques is designed, implemented and tested. Experiments with the application are performed in the Uninett network, with the goal of identifying the impact of different network properties on geolocation.

What most clearly set this thesis apart from earlier work, in addition to the use of a multi-agent system, is the analysis of the impact of IPv6 on geolocation, and the introduction of multi-party computation to geolocation. The extensive focus on delay measurements, although not bringing anything new to the field of networking in general, is also new to geolocation as far as we know.

Keywords: Internet forensics, multi-agent systems, geolocation.
Preface

This Master’s thesis is the result of the 10th semester of my master’s program at the Department of Computer and Information Science at the Norwegian University of Science and Technology.

The outline for the assignment was proposed by Espen André Fossen at the High Tech Crime Division of the National Criminal Investigation Service (KRIPOS) and André Årnes at the Center for Quantifiable Quality of Service in Communication Systems (Q2S). As supervisor André Årnes helped flesh out and define the final assignment.

I would like to thank André Årnes and supervising professor Svein Johan Knapskog for valuable input and feedback. Additionally I would like to thank PhD student Tord Ingolf Reistad for his help with the theory of multi-party computation, and Jon Kåre Hellan and Morten Knutsen at Uninett for their quick and to the point response to any problems regarding the Uninett network infrastructure used in this project. Special thanks goes to Hans Christian Falkenberg at Fast Search & Transfer for sharing his considerable knowledge of the Java programming language, and for identifying many corner-cases in the implementation of algorithms used and developed in this work.

Trondheim, November 15 2006

Øystein E. Thorvaldsen
# Contents

1 Introduction ............................................................... 1  
  1.1 Motivation .......................................................... 1  
  1.2 Background .......................................................... 1  
  1.3 Purpose and Goals .................................................. 2  
  1.4 Limitations .......................................................... 2  
  1.5 Document Organization ............................................ 3  

2 Digital and Internet Forensics ........................................ 5  
  2.1 Introduction .......................................................... 5  
  2.2 Internet Forensics ................................................... 5  
  2.3 The Chain of Custody ............................................... 6  

3 Geographical Location of Internet Hosts .............................. 7  
  3.1 Introduction to Geolocation ....................................... 7  
  3.2 Different Approaches to Geolocation .............................. 8  
    3.2.1 Using Public Information Sources .......................... 8  
    3.2.2 Measurement-based Approaches ............................... 11  
    3.2.3 Distance Maps ................................................ 12  
  3.3 Related Work ........................................................ 12  
  3.4 Comparative Analysis ............................................. 14  
    3.4.1 Common Limitations ............................................ 14  
    3.4.2 Using Public Information Sources .......................... 17  
    3.4.3 Measurement Based ............................................. 22  
  3.5 Improvements to Current Techniques .............................. 25  
    3.5.1 Combining Information Sources and Measurements .......... 26  
    3.5.2 Dynamic Regions and Super-Landmarks ....................... 26  
    3.5.3 Limited Knowledge of Landmark Locations ................... 27  
  3.6 Delay Measurement ................................................ 30  
    3.6.1 Delay Components .............................................. 30  
    3.6.2 Ways to Measure Delay ....................................... 31  
    3.6.3 Confidence in Delay Measurements ........................... 34  
    3.6.4 From Delay Measurements to Geographical Distance ....... 36  
  3.7 Internet Protocol v6 ............................................... 36
4 Geolocation in MAFIF

4.1 The Existing MAFIF Framework

4.2 Geolocation Algorithms

4.3 Delay Measurements

4.4 User-Interaction and Management

4.5 Geographical Functionality

4.6 Address Information Storage

4.7 Limitations

5 Experiments

5.1 Environment - The Uninett Network

5.2 Test Setup

5.3 Limitations

5.4 Experiments and Results

6 Conclusions
7 Further Work

7.1 Large-scale Experiments ................................................. 81
7.2 Multi-Party Computation ............................................... 81
7.3 IPv6 ............................................................................. 82
7.4 Detection of Direction of Movement ..................................... 82
7.5 Web Service Integration .................................................. 82

References ........................................................................... 82

A Article .................................................................................. 95

B Design Diagrams ................................................................. 107
  B.1 (A)UML Diagrams .......................................................... 108
      B.1.1 AdminAgent ......................................................... 108
      B.1.2 SessionAgent ..................................................... 109
      B.1.3 WorkerAgent ..................................................... 111
      B.1.4 GWAgent ........................................................ 113
  B.2 Servlet UML Diagrams .................................................... 115
      B.2.1 Servlet Core Classes ............................................. 115
      B.2.2 Servlet Alpha Classes ......................................... 116
      B.2.3 Servlet Circle Classes ......................................... 117
  B.3 Database ........................................................................ 118
      B.3.1 UML Class Diagram ............................................. 118
      B.3.2 Database Tables ................................................ 119

C Source Code ......................................................................... 121
  C.1 AdminAgent Classes ..................................................... 121
      C.1.1 AdminReplyGWBehaviour .................................... 121
      C.1.2 GWCase ......................................................... 125
  C.2 SessionAgent Classes .................................................... 127
      C.2.1 InitiateTraceSessionBehaviour ............................... 127
      C.2.2 TraceReduceBehaviour ....................................... 135
      C.2.3 DelayVector ...................................................... 143
  C.3 WorkerAgent Classes .................................................... 145
      C.3.1 WorkerTraceStart ............................................... 145
      C.3.2 WorkerTraceLandmarks ...................................... 151
      C.3.3 WorkerTraceCalculate ....................................... 152
      C.3.4 WorkerTraceTarget ........................................... 157
      C.3.5 WorkerTraceFinal .............................................. 160
      C.3.6 PingBehave ....................................................... 164
  C.4 GWAgent Classes .......................................................... 166
      C.4.1 GWAgent ........................................................ 166
      C.4.2 GWReceiveBehaviour ......................................... 170
      C.4.3 LaunchTraceBehave ............................................ 173
List of Figures

3.1 Excerpt of a RIPE IP whois reply ........................................ 9
3.2 Excerpt of a Norid DNS whois reply ................................. 10
3.3 A BGP table entry .......................................................... 11
3.4 Safety margin in CBG ...................................................... 23
3.5 Selection by super-landmark ............................................. 27
3.6 CBG as grid coverage ..................................................... 28
3.7 Limited Multi-Party Computation ....................................... 29
3.8 Components of Network Delay ......................................... 31
3.9 Congestion Regions ....................................................... 35
3.10 MIP Triangle Routing ..................................................... 38

4.1 MAFIF Design ................................................................. 39
4.2 Sequence Diagram for a Trace .......................................... 41
4.3 TraceBehaviour Logic ..................................................... 42
4.4 Scatter plot of distance and delay ..................................... 43
4.5 Sphere trigonometry ........................................................ 45
4.6 Exact area of intersecting circles .................................... 45
4.7 Ping outputs .................................................................... 47
4.8 Ping class relations ........................................................ 48
4.9 JADE Gateway ............................................................... 49
4.10 GUI program flow .......................................................... 50
4.11 Servlet class relations .................................................... 51
4.12 Data model Landmarks table .......................................... 56

5.1 Uninett Network Topology ................................................ 60
5.2 Uninett measurement nodes .............................................. 61
5.3 Distance between measurement nodes ............................... 62
5.4 One-Way Delays .............................................................. 63
5.5 One-Way RTT Correlation ............................................... 64
5.6 Comparison of minimum RTTs with varied probe parameters 66
5.7 Comparison of minimum RTTs and C values ...................... 67
5.8 Comparison of minimum RTTs with downed link ............... 68
5.9 Differences between IPv4 and IPv6 in CBG ....................... 70
5.10 Confidence region for trd-mp ........................................... 72
5.11 CBG confidence region for bo-mp ........................................ 72
5.12 CBG underestimation ....................................................... 74
5.13 CBG runs for moving target .............................................. 75

B.1 AUML diagram AdminAgent ............................................. 108
B.2 AUML diagram SessionAgent ........................................... 109
B.3 AUML diagram SessionAgent’s Behaviours .......................... 110
B.4 AUML diagram WorkerAgent ............................................ 111
B.5 AUML diagram WorkerAgent’s Behaviours ......................... 112
B.6 AUML diagram GWAgent ................................................. 113
B.7 AUML diagram GWAgent’s Behaviours .............................. 114
B.8 UML diagram servlet core ............................................... 115
B.9 UML diagram servlet alpha ............................................. 116
B.10 UML diagram servlet circles .......................................... 117
B.11 UML diagrams of Database classes ................................. 118

D.1 UTM Zone discrepancy ..................................................... 273
# List of Tables

3.1  Comparison of Geolocation Techniques ................. 15
5.1  Differences in delay between IPv4 and IPv6 ............... 70
5.2  GeoPing results using all measurement nodes ............... 71
5.3  GeoPing results using southern measurement nodes ........... 71
5.4  CBG results. ........................................... 72
5.5  Moving target GeoPing results ...................... 76
5.6  Δ(DV) variances for multiple GeoPing runs ................. 78
Chapter 1

Introduction

This thesis presents a multi-agent system for determining the geographical location of Internet hosts. The system expands the multi-agent framework for Internet forensics presented in our minor thesis [1]. As such the focus will be on the geo-location functionality, and not on the underlying framework. A comprehensive theoretical introduction to geolocation and related subjects is provided as a basis for the implementation. This chapter presents the motivation and background for our work, what we hope to accomplish and the limitations that specify this work.

1.1 Motivation

There is currently no international body of laws that govern acts of digital crime where the crime scene spans multiple countries. The many national laws that more or less cover digital crime are not harmonized. What constitutes a criminal act in one country might not be illegal in another. This makes it important to be able to determine the location(s) of any investigated actions, to aid law enforcement in contacting the relevant authorities and to apply the correct body of laws. Actually locating the area where the source(s) of a criminal act might be located physically may also help law enforcement in seizing important evidence and detaining suspects.

1.2 Background

The High Tech Crime Division (HTCD) at the National Criminal Investigation Service (Kripos) is responsible for digital forensics work. The HTCD perform its own investigations but also acts as a national resource and knowledge center in this matter. Part of this responsibility is to keep up to date on new technologies, techniques, threats and trends in digital forensics. The pace of development in
the field is high, and keeping the equipment and personell at HTCD up to speed requires a lot of resources. To address some of these challenges the HTCD has over the last years co-operated with the department of telematics (ITEM) at NTNU to promote research in the area of digital forensic science.

HTCD contributes assignments and external examiners for ITEM MSc. students that specialize in information security. In return HTCD gets full access to the resulting work, and can concentrate more internal resources on other pressing matters. To date this cooperation has resulted in the following work in the field of Internet forensics: [2, 3, 4, 1].

During this work the basics of digital forensic science theory, terminology and practice have been established. As such delving into a broad description of the field in this thesis would not serve any purpose. We have however included a short introduction to digital and Internet forensics that should be comprehensive enough to make the thesis self-contained in this aspect.

1.3 Purpose and Goals

Currently several techniques for performing geographical location of Internet hosts exist, but none are very accurate. We compare some of these techniques, and try to determine if better knowledge of the network topology and conditions improve the quality of the results to a degree where the effort needed to acquire this knowledge can be justified.

Particularly we look at the following:

- Differences between IPv4 and IPv6 relevant to geolocation.
- Correlation between one-way delay, round trip time and geographical distance.
- Challenges to and techniques for capturing as precise delay measurements as possible.
- Comparison of current geolocation techniques, and possible improvements.

1.4 Limitations

Geographical location of Internet hosts has many purposes other than those of Internet investigation, such as targeted advertising and language selection for websites. We do not take these into account when discussing the different methods, and no experiments are performed to assess the suitability of any techniques for such purposes. More specific limitations regarding the design and implementation
of the geolocation functionality in the Internet forensics framework is described in Section 4.7.

1.5 Document Organization

This chapter presented the motivation for our work, what we hope to accomplish and the approach and limitations that specify it. The rest of the thesis is divided into 6 chapters as follows:

- Chapter 2 gives a brief overview of the disciplines of digital and Internet forensics, and how they relate to the rest of the work in this thesis.
- Chapter 3 gives an introduction to geographical location of Internet hosts, discusses and evaluates previous work and puts forth suggestions for improvements.
- Chapter 4 describes the design and implementation of geolocation functionality as an application in the existing multi-agent framework.
- Chapter 5 details the test environment, the different experiments and their results.
- Chapter 6 summarizes the work and draws conclusions.
- Chapter 7 suggests areas for further work.

The use of plural references to self in this thesis is just a form of expression, and does not indicate any involvement of external parties other than the normal process of supervision.
Chapter 2

Digital and Internet Forensics

A brief introduction to the disciplines of Digital Forensics and Internet Forensics is given here to provide a context for geographical location of Internet hosts.

2.1 Introduction

Digital forensics is a specialized part of forensics that deals with the securing and handling of digital evidence. Internet forensics is a sub-discipline of digital forensics that deals with the securing and handling of digital evidence on the Internet. Digital forensics can be defined as:

The use of scientifically derived and proven methods toward the preservation, collection, validation, identification, analysis, interpretation, documentation and presentation of digital evidence derived from digital sources for the purpose of facilitating or furthering the reconstruction of events found to be criminal, or helping to anticipate unauthorized actions shown to be disruptive to planned operations [5].

This definition requires us to also define the term "digital evidence". We will use the definition of the International Organization on Computer Evidence:

Any information stored or transmitted in binary form that may be relied upon in court [IOC06].

2.2 Internet Forensics

Internet forensics differs from digital forensics mostly in its narrower scope and more problematic access to evidence. Unique to Internet forensics is that investigators may have access to a crime scene without knowing its geographical locat-
ion(s). This means that determining the location(s) becomes an important part of an investigation.

A more thorough discussion of the relations between digital and Internet forensics is available in [1], where a terminology covering the most important concepts of digital forensics is also presented. Different models and frameworks for conducting investigations involving digital and Internet forensics are compared in [6].

2.3 The Chain of Custody

The chain of custody is one of the most important principles in all of forensics. The principle can be summarized as: An identifiable person must at all times have the physical custody of a piece of evidence. This means a qualified person like a police officer will take charge of it, document its collection, and hand it in for storage in a secure place. These transactions, and every succeeding transaction between the collection of the evidence and its appearance in court, must be completely documented in order to withstand challenges to the authenticity and integrity of the evidence. Documentation should include the conditions under which the evidence is gathered, the identity of all evidence handlers, duration of evidence custody, security conditions while handling or storing the evidence, and the manner in which it is transferred to subsequent custodians each time such a transfer occurs [ccW06].

When dealing with digital evidence this principle is extremely important, as tampering is much easier than with traditional evidence, and more likely to go unnoticed [7]. There is no single way to enforce chain of custody in digital forensics, but the use of techniques such as time-stamping and hashing algorithms are central to all methods. Digital signatures would offer increased security but is currently not widely used in this context.

With regard to geographical location of Internet hosts the principle makes the following information important: When was a location determined, how was it done, who participated and what information did they contribute, and finally who ordered that an operation to determine the location should be carried out. In addition to this the result must be secured sufficiently to protect against any malicious or accidental alteration.
Chapter 3

Geographical Location of Internet Hosts

In this chapter we introduce the problem of geographical location of Internet hosts (geolocation). Different classes of techniques for doing this are identified, and we describe related work and existing techniques within the context of these classes. Delay measurement is at the core of several of these techniques, and we go into depth describing the challenges of accurately measuring delay. The Internet is in a slow transition to IPv6, we point out any effects the use of IPv6 might have on geolocation compared to the current IPv4. We also compare the strengths and weaknesses of the existing techniques and propose possible enhancements.

3.1 Introduction to Geolocation

There are many possible ways to determine the geographical location of an Internet host, the simplest might be to just look up the alleged assignee and ask him or her. However, we will consider only technical solutions, and as we will see later, the information registered about the alleged assignee of any particular IP address or DNS entry might not be that accurate and trustworthy anyway. Thus the question becomes how to determine the location of a host with the least effort and the most accurate result, in a way that can be automated.

Geolocation has many possible applications, we look primarily at its use as a forensic tool. As such geolocation should be seen as an integral part of Internet forensics, and its application should be incorporated into an overall investigation model. We will not go into a discussion of digital forensic models and frameworks here, as we focus on the technical aspects of geolocation. It is, however, important to keep forensic principles such as the chain of custody in mind, and we will
take such considerations into account in our implementation and experiments in Chapters 4 and 5.

Narrowing the scope to a forensic application leads to a different set of criteria than if we were to consider geolocation for general purposes, such as location dependent content or advertising. The massive scalability usually required on the Internet for publicly available services will for instance not be necessary. Also cost considerations, the need for special equipment and access to information can be treated differently. Some of the techniques used for general purpose geolocation can also be applied in a forensic context, the main problem in doing this is the accuracy of the results, and not least whether they can be trusted. An assessment of the suitability of different techniques is performed in Section 3.4.

3.2 Different Approaches to Geolocation

Two main classes of approaches to geolocation can be identified. Approaches relying on publicly available information sources that does not at all actively query the host in question, and approaches that try to infer the location of a host using measurements. A third somewhat hybrid approach; using different sorts of pre-calculated distance maps can also be identified. Combinations of these different approaches are of course possible, and we consider this in Section 3.5.

3.2.1 Using Public Information Sources

Fossen goes into great detail about using public information sources for geolocation in [2]. We therefore only briefly describe some of the different sources here.

IP whois

Originally defined in [RFC812] and later updated in [RFC954, RFC3912] the whois-service provides a mechanism for finding contact and registration information for Internet resources. The current service is structured by Top Level Domains (TLD) or Country Code Top Level Domains (ccTLD).

Five Regional Internet Registries (RIRs) administer the allocation of IP addresses on behalf of the Internet Assigned Numbers Authority (IANA). The registries’ databases typically contain IP addresses, Autonomous System numbers and organizations or customers that are associated with these resources. The RIRs again delegate the allocation of addresses in their regions to Internet Service Providers and other organizations [who06], [RFC2050, RFC1918, RFC3330].
Information about a particular IP address can be obtained by querying one of the RIRs. Only the American RIR (ARIN) has information about which RIR an address is managed by. A first query should be directed at ARIN, which will either contain a record for the address or a pointer to which RIR that does. Replies to queries are in the Routing Policy Specification Language (RPSL)\(^1\) defined in [RFC2622], an update also covering IPv6 is defined in [RFC4012]. Figure 3.1 shows part of a typical query result, listing country, city and even street address.

**Figure 3.1**: Excerpt of a reply from RIPE about IP address 129.241.190.190.

**DNS whois**

The whois service can also be used for querying Domain Name System (DNS) records. DNS was introduced in [RFC882, RFC883]. A DNS record is in its simplest form a mapping from a computer host name to an IP address. The mappings are all registered in the worldwide DNS. The DNS is a hierarchic system that in its current revision divides the name space into TLDs and ccTLDs [RFC920, RFC1034, RFC1035]. Due to this hierarchic structure there are no central regional registries like the ones for IP addresses, queries are processed along the hierarchy. Most TLDs and ccTLDs make whois databases with information about the registrants publicly available. Unfortunately there is no common format like RPSL for these databases. With at least 14 TLDs and over 100 ccTLDs, the task of automating DNS whois queries and interpreting the replies correctly involves a lot of work. However, DNS records can contain additional information to that available about the IP addresses they map to\(^2\) [dns06b]. Additionally [RFC1712, RFC1876] propose adding geographical information to DNS records, although

\(^1\)RPSL is rather complex, and the RIRs use different “dialects” making it necessary to tailor any automated query program to the different RIRs.

\(^2\)Several DNS records may map to the same IP address, but the registrant information may differ between the records.
this has not been widely adopted. Figure 3.2 shows part of a query result from the Norwegian ccTLD Norid for the domain ntnu.no. Again country, city, street address and other contact information is listed.

Domain Name................: ntnu.no
Organization Handle........: NTU10-NORID
Registrar Handle..........: REG2-NORID

Additional information:
Created: 1999-11-15
Last updated: 2005-08-26

Organization Name.........: Norges Teknisk-Naturvitenskapelige Universitet
Post Address...............: Høgskoleringen 1
Postal Code................: N-7491
Postal Area................: Trondheim
Country....................: Norway
Phone Number...............: +47 73 59 50 00

Figure 3.2: Excerpt of a reply from Norid about ntnu.no.

DNS names can also be used directly to infer geographical location. Many network providers name their routers according to some internal geographical naming convention. There is no standard convention, so this approach requires tuning for every network operator. If the geographical location of the last hop router can successfully be inferred it is reasonable, due to the structure of the Internet, to assume that the host in question is within a limited distance from this router.

Routing Information

Both IP and DNS records may reveal where a host is supposed to be. Routing information on the other hand might show where traffic destined for a particular host actually travels. This is possible due to the Internet’s use of route publishing. Route publishing is the dissemination of reachability information. That is, where to send packets for them to reach their intended destination. The Border Gateway Protocol (BGP) is the protocol used for this between Autonomous Systems on the Internet [RFC4271]. An Autonomous System (AS) is defined in [RFC1930] as:

A connected group of one or more IP prefixes run by one or more network operators which has a SINGLE and CLEARLY DEFINED routing policy.\(^3\)

\(^3\)The classic definition of an Autonomous System is a set of routers under a single technical administration with a single internal and single external routing policy. The updated definition takes into account that only the externally presented picture of what networks are reachable through the AS is important.
A unique AS number (ASN) is allocated to each AS by IANA for use in BGP routing.

It is possible to query ASes for the network prefixes they route, and thus get an estimation of the path travelled and the final destination of a particular traffic flow, based on the geographic area covered by the ASes and the information registered at the RIRs about them. An example is shown in Figure 3.3.

BGP routing table entry for 129.240.0.0/15, version 24473688
Bestpath Modifiers: always-compare-med, deterministic-med
Paths: (2 available, best #1)
  Advertised to update-groups:
    1
  224 193.10.68.1 (metric 11) from 193.10.68.1 (193.10.68.1)
     Origin IGP, metric 0, localpref 131, valid, internal, best
     Community: 2603:111
  224 128.39.0.89 from 128.39.0.89 (128.39.0.89)
     Origin IGP, metric 91, localpref 129, valid, external
     Community: 2603:111

Figure 3.3: The BGP table entry for 129.241.0.0 at no-gw2.nordu.net, showing ASN 224 as the destination. No AS path is shown as no-gw2.nordu.net has a direct route to ASN 224.

3.2.2 Measurement-based Approaches

As opposed to the approaches described above, measurement-based approaches may produce network traffic to the target host, depending on how the measurements are performed. There are two main types of measurement-based approaches; active and passive.

Active Measurements

Active measurements probe the target host thus generating network traffic. The type and amount of probing varies depending on the technique used. Common to all the techniques is that one attempts to find the delays between the target host and several probing machines, called landmarks, with known locations. This requires that the target host actually replies to probe requests, for more on this see Section 3.6. The delay values gathered are then used to calculate the approximate location of the host, either by doing an analysis of the generated delay pattern or by translating the delay measurements into geographical distance. A description of such techniques is given in Section 3.3.
Passive Measurements

Using special equipment, in the form of passive measurement cards, it is possible to measure and analyse traffic without affecting the network traffic at all [8]. For such techniques to be successful it is necessary that the target host itself generates traffic that passes through the network where such equipment is installed. If such techniques are to be generally useful an extensive network of passive measurement equipment must be deployed. The European Union projects A Scalable Monitoring Platform for the Internet (SCAMPI) and Large-Scale Monitoring of Broadband Internet Infrastructure (LOBSTER) projects have deployed such equipment on parts of the European backbone [sca06, lob06].

It is also possible to capture traffic generated by the target host at higher protocol levels, and calculate delays based on this. Muir et al. suggests using HTTP-refresh for estimating RTT to target hosts [9]. Techniques based on HTTP-refresh or similar concepts are not strictly passive. Even though it is the target host that initiates the traffic the host(s) trying to locate the target will generate subsequent traffic to the target host.

In wireless networks the signal strength may be measured and triangulated. This is outside the scope of this project.

3.2.3 Distance Maps

A distance map is a representation of perceived distances between hosts, irrespective of their geographical location, where the distances are measured as network delay. Many schemes for creating distance maps for (parts of) the Internet has been proposed, see Section 3.3. Depending on the techniques used to create and maintain the map and respond to queries, the approach to some extent uses active measurements. Typically active measurements between all or some hosts are needed to create the initial map, while queries are answered from already assembled information, making distance map based approaches more or less hybrids between using existing information sources and performing measurements.

3.3 Related Work

Padmanabhan and Subramanian introduced GeoPing in [10]. This is to our knowledge the first measurement-based technique for geographical location of Internet hosts. Manual use of traceroute, ping and several techniques for extracting and compiling information from public information sources, such as DNS records and IP whois, of course precede this work.
GeoPing works by building a map $M$ of delay vectors. Each vector represents the delay to a single host with known location from a set of probes $N$, also with known locations. A delay vector $DV$ for a target $T$ with unknown location is then constructed by measuring the delay from all probes in $N$ to $T$. $DV$ is then compared to every vector in $M$ to find the closest match. This is done by considering the vectors in $M$ as an $N$-dimensional delay space, and calculating the Euclidean distance between $DV$ and every other vector. The "nearest" neighbor to $DV$ is returned as the location estimate of $T$.

The principle behind GeoPing has been refined in [11, 12] by Ziviani et al by introducing different similarity models for calculating which host exhibits the closest matching delay pattern. In [13] a further refinement, placing probes according to population density is suggested. The idea is to improve results with fewer probes, and avoid overlapping measurements. Guye et al improve upon this idea by introducing a two-tiered approach in [14]. An upper level handles long distance measurements, and a lower level keeps measurements within restricted areas.

GeoPing-based techniques have an important shortcoming. The result of a location attempt is a discrete set of possible locations, limited to the hosts participating in the location process. Constraint-based geolocation (CBG) introduced by Guye et al in [15] addresses this through the use of multilateration, and provides a location with a continuous confidence region as its estimation result. The set-up is much as in GeoPing, but delay measurements are converted into actual geographical distances. For each landmark $L_i$ CBG calculates a best-line $b_i$ based on delay measurements between $L_i$ and every other landmark $L_{j\neq i}$. The best-line represents the least distorted relationship between the measured delay and the actual geographic distance for each landmark. $b_i$ is then subtracted from the delay measurement between $L_i$ and $T$, for all $i$. The results are then converted into geographical distance constraints used to multilaterate the location of $T$.

Fossen implemented a CBG-based system for western Europe in [6, 2], using publicly available Looking Glass hosts as landmarks. He also discussed the use of publicly available information sources.

Although providing a continuous confidence region, and to some extent mitigating measurement distortion, the original CBG still did not give a very exact location of the target host. Guye et al improved upon their earlier work by estimating the buffer delay part of the total delay used in their computations in GeoBud [16]. Delays adjusted for buffer delay results in smaller confidence regions, and thus less error in location estimation. Compared to CBG the introduction of buffer delay estimation in GeoBud improved results by about 27% for hosts located in Western Europe and by about 37% for hosts in the United States, for the datasets used. This higher accuracy come at the cost of geolocating routers along all relevant paths and measuring their buffer delay.

Several schemes for a publicly available infrastructure for measuring network distance
between hosts have been proposed. Although these schemes do not seek to locate
hosts geographically, but rather construct a distance-based map of the Internet for
proximity-purposes, they do provide valuable information about the distribution
of Autonomous Systems, possible simplifications and how they influence the acc-
uracy of the results. The first such scheme was IDMaps [17] by Francis et al, later
improved upon in [18, 19, 20, 21]. More recent work has focused on distributing
the load, reducing network traffic and determining accuracy over time [22, 23, 24,
25, 26]. These schemes are primarily meant for selection of lowest latency servers
or peers in general applications, and are currently not accurate enough for forensic
needs.

As far as we can determine, no previous work exists that take into account the
impact of IPv6 on geolocation techniques. As in [14], we propose a tiered approach,
but do this in a more dynamic way, by not operating with two strictly separate tiers.
By using multi-agent technology we are able to invest more advanced behaviour
into our landmarks, distribute the load of computation and integrate geolocation
functionality into a general Internet forensics platform, such as that in Appendix
A. Also we believe our suggestions for how to use multi-party computation in
geolocation are novel.

3.4 Comparative Analysis

So far we have not discussed advantages and drawbacks to the different approaches
and techniques. Here we go into detail about accuracy, trustworthiness, required
effort, sources of error and possible circumventive acts for each of the techniques.
A summary is given in Table 3.1, where H indicates high, M medium and L low
scores. Note that for the two categories Detectability and Effort a high score is
not positive. The Geocluster and GeoBud techniques as well as inference based on
DNS names are evaluated independently to highlight their differences from related
techniques.

3.4.1 Common Limitations

Before going into detail about each technique some common limitations are im-
portant to keep in mind. In addition to the problem areas described below Mobile
IP may also affect the results. The possible impact of Mobile IP is discussed in
Section 3.7.
Table 3.1: Summary of Comparison of Geolocation Techniques. * Note that IDMaps performs delay estimations not geolocation. See the analysis of IDMaps in Section 3.4.3.

<table>
<thead>
<tr>
<th>Technique</th>
<th>Accuracy</th>
<th>Detectability</th>
<th>Freshness</th>
<th>Reliability</th>
<th>Effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>whois IP</td>
<td>M/L</td>
<td>L</td>
<td>M</td>
<td>M</td>
<td>L</td>
</tr>
<tr>
<td>whois DNS</td>
<td>M/L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>M/L</td>
</tr>
<tr>
<td>Routing Info</td>
<td>M</td>
<td>L</td>
<td>H</td>
<td>H</td>
<td>M/L</td>
</tr>
<tr>
<td>DNS names</td>
<td>M</td>
<td>L</td>
<td>M/H</td>
<td>L/M</td>
<td>M</td>
</tr>
<tr>
<td>GeoCluster</td>
<td>M</td>
<td>L</td>
<td>M</td>
<td>M/L</td>
<td>M</td>
</tr>
<tr>
<td>IDMaps*</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>M/L</td>
<td>M/H</td>
</tr>
<tr>
<td>GeoPing</td>
<td>H/M</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>M</td>
</tr>
<tr>
<td>CBG</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>M</td>
</tr>
<tr>
<td>GeoBud</td>
<td>H⁺</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
</tr>
</tbody>
</table>

**Slow Links and Congestion**

Slow links results in large delays, if the distribution of slow links in the network(s) travelled by probe packets is not relatively uniform the results may be skewed. Congestions can make links appear as slow, but may be detected using the technique described in Section 3.6.3.

**Topology-Hiding**

The result of a tracing operation might be correct, without being of much value. This is due to the use of different topology hiding techniques such as proxies, Network Address Translation (NAT) and Virtual Private Networks (VPN). In most cases the use of these techniques are legitimate, but they can also be used for intentionally making a host difficult or impossible to trace.

**Proxies** A proxy server is a host that offers a network service to allow clients to make indirect network connections to other network services [wik06b]. With regard to geolocation the most important feature of a proxy is that the address of the real source is hidden, it is the address of the proxy that is publicly visible. Thus the address left behind by a target using a proxy will be the address of the proxy. Tracing this address then will if successful give the location of the proxy. If the proxy is local to a company, school or some other organization this might not be a problem. At least not insofar as finding the geographical location of the source. If on the other hand the proxy is open the location of the proxy itself might be worthless. An open proxy can be defined as: "a proxy server which will accept client connections from any IP address and make connections to any Internet
Law enforcement may be able to seize the open proxy and get the real source addresses from it. This becomes practically impossible if several open proxies are chained to create a path of anonymity. Chaum introduced the concept of a mix-network, a set of servers that serially encrypt or decrypt incoming messages and outputs them in a random order, so that an outsider cannot correlate input and output messages [27]. Several schemes inspired by this concept to hide original source addresses have been proposed, and some are in use on the Internet. A comprehensive list of publications is available in [ano06]. The best known and most used is probably The Onion Routing (TOR) network [28].

**Network Address Translation** NAT, also known as network masquerading or IP-masquerading involves re-writing the source and/or destination addresses of IP packets as they pass through a router or firewall. The original purpose of NAT was to enable multiple hosts on a private network to access the Internet using a single public IP address [wik06a].

From the point of view of geolocation NAT works in about the same way as a proxy, hiding the original source address. Some NAT-devices can be configured to forward incoming request to hosts inside the NAT, and as such allow direct connections, but the source address will still remain hidden.

**Virtual Private Networks** VPN is a set of techniques used to communicate confidentially over a publicly accessible network by constructing a virtual network on top of the publicly available infrastructure and protocols, for instance the Internet [wik06c].

A client participating in a VPN configured so that all IP traffic passes through the VPN tunnel seems to not exist to other hosts, only the entry point to the VPN is visible. This entry point might be at a totally different location than the client host(s).

**Temporary Addresses**

Traditionally many users connected to the Internet used dial up connections. This gave the user a new IP address each time she connected. With broadband connections becoming more common, the number of dial up users are falling, but many broadband connections also routinely change the IP addresses of their clients. Publicly available wireless hot-spots also provide their users with temporary addresses. Thus an address might be in use by someone else (possibly at a different location) than the intended target at the time the trace is being performed.

---

4Wireless hot-spots may employ different technologies, not all hot-spots provide their users with public IP addresses, thus also working as NAT-devices or proxies.
3.4.2 Using Public Information Sources

All use of public information sources have the drawback that the information is at some point submitted by the registrant. The information might also be dated. The greatest advantages to using public information sources are without doubt that no traffic is generated to the target host, and that all that is needed is a simple query and interpretation of the reply, no landmarks or calculations are necessary.

IP whois

**Reliability** It is quite costly to be assigned a range of IP addresses, and the information about owners required by the RIRs is comprehensive. It is of course possible to falsify this information, but as most IP range owners are major corporations or the like it would probably not be in their interest to do so. Also, the RIRs or their sub licensers are likely to actually use the provided information to contact the alleged owners, leading to a greater possibility of detecting false or erroneous information. The RIRs may also take down the address space for investigation if it is unused or not set up correctly.

An IP whois record can be hijacked\(^5\). That is, the record can changed by an unauthorized individual posing as the legal assignee. IP hijacking can be done in several different ways, which we will not go into here. An introduction is given in [hij06b]. The portion of the total address space being in a hijacked state at any time is low. However, as most hijackings are the result of criminal intent (only a small portion is due to mis-configuration), it is not improbable that addresses from hijacked ranges will be overrepresented in law enforcement cases where geolocation could be useful. A relatively up to date list of suspected and confirmed hijacked address ranges is available at [hij06a].

**Accuracy** It is the assignee’s contact information that is required in the registration, not where the owner chooses to actually deploy the addresses. This might lead to erroneous assumptions about the location of hosts using the addresses. Also, if the assignee is an organization with operations at different locations, or the assigned range is large, parts of the range is likely to be deployed at locations different from the one registered.

**Freshness** IP whois records may have a field specifying when the information was last updated. This is not the case for the record in Figure 3.1. As two arbitrary examples the update field for the range 18.0.0.0/8 assigned to Massachusetts Institute of Technology was last updated September 26 1998, while the range 207.46.0.0/16

---

\(^5\)Not to be confused with IP spoofing, the sending of IP packets with false header information.
assigned to Microsoft Corporation was updated December 9 2004. However, the somewhat elaborate registration process necessary to be assigned an IP range leads to changes in assignment being rather infrequent. As long as the registration information has not been falsified it is reasonable to assume that it is also up to date.

**Detectability** The possibility of a target detecting that someone is querying RIRs about its registration details is practically non-existent. If the target, hypothetically, has the capability to run sufficiently extensive surveillance to detect such attempts it would undoubtedly be within its capacity to out-smart any attempts to trace it at all.

**Effort** There are two hurdles of any difficulty worth mentioning in this regard. The first is that to automatically extract useful information from the reply to an IP whois query one must take into account the different RPSL syntaxes used, and build a database to match the extracted information against. The second is that the different whois services may limit the number of connections from an address/host in a given period of time, resulting in the need for a pool of addresses to use for querying. Both these hurdles are very manageable, compared to the challenges associated with the other approaches.

**DNS whois**

**Reliability** DNS records may, as IP range records, contain falsified information. However, as registering a domain name is a much simpler process, and the number of DNS registrants is much higher than for IP ranges it is much easier to supply incorrect information. Also this information is less likely to be validated, as the number of DNS records is much higher and their importance much lower compared to IP range records.

Just as with IP range records it is possible to hijack DNS records. The effect is different though. By modifying a DNS record one can redirect any requests to another host. This is often used to redirect unsuspecting users to fake pages created by attackers. This is not of interest to us. However an attacker might change DNS records to confuse investigators by pointing them to hosts not involved in the investigated actions. The IETF is in the process of developing standards for solving different security problems related to DNS, but these are currently not widely deployed or not finished [RFC4033] [dns06a].

**Accuracy** As with IP range records it is the registering organization/person’s contact information that is required in the registration. The registrant is technically free to point the DNS record to any host on the Internet, and is more likely to do so than in the case of IP ranges.
Freshness  DNS records have a field specifying when the information was last updated, as can be seen in Figure 3.2. Domains change hands and are abandoned regularly. It is not uncommon for the contact information listed in DNS records to be outdated. To assume that the information is up to date the update field should indicate that the information was changed relatively recently.

Detectability  The possibility of a target detecting that someone is performing queries about its registration details is higher than for IP ranges, due to the hierarchic nature of the DNS. A request for information about a particular address will be forwarded to the DNS server(s) responsible for the record, and this might be controlled by the person in control of the targeted host. Still, to filter out such queries from other DNS request requires knowledge of the tracing operation and persistent monitoring.

Effort  The problem of limited access does not apply to DNS queries to the same extent as for IP whois. Different DNS servers may have different policies, and as there are many more of them successive queries are more likely to be to different servers. The problem of extracting the information from replies is on the other hand more demanding, due to missing common formats and the large number of servers. The formats are also likely to change more frequently than for IP range records, resulting in a higher maintenance effort.

Routing Information

Reliability  Routing information needs to be correct for the network to work at all. ASes are generally run by large organizations which either depend on the routing to work for their own operations, or they act as Internet Service Providers and sell access to customers, who also need the routing to work.

Accuracy  An AS can be very large. Many network providers are transnational companies, and depending on their internal policies they may employ from one to many ASes. As such an AS can cover a large geographical area. Also the information registered about the assignee might be for some sort of central office, and not the local branch of the organization.

Accuracy may be increased by combining AS lookups with inference based on DNS-naming of routers, but this is a technique fraught with error sources.

Freshness  Inter-AS routing is policy based, with deals between the different ASes, and is as such rather stable. However, due to downed links or other network
problems temporary route changes may happen more rapidly. Thus the AS path actually travelled may vary, but the final destination is the same.

**Detectability**  The possibility of a target detecting that someone is querying ASes about its registration details is practically non-existent. If the target, hypothetically, has the capability to run surveillance extensive enough to monitor enough ASes to detect such attempts it would no doubt be within its capacity to out-smart any attempts to trace it at all.

**Effort**  If one is interested in information about the ASes on the path to the final destination successive queries might be necessary. The biggest problem is probably one of access. Earlier BGP routing information was generally openly published. Due to security concerns more and more ASes limit the availability of this information. This is the case for Uninett which previously published BGP information freely on its web site, but now only makes this available to selected partners.

**DNS names**

Note that to infer locations based on router names it is necessary to first establish which routers are on the route to the host, and as such this technique can not be used independently.

**Reliability**  There is no guarantee that a router name is based on geographical location. Even if this seems to be the case it might not be, and lead to false conclusions.

**Accuracy**  If the geographical location of the last hop router to the target can successfully be inferred from its name it is reasonable to assume that the target host is within a limited distance from this router. Depending on the network density what constitutes a limited distance might vary considerably, and this must be accounted for.

**Freshness**  If a router name is based on its geographical location it is natural to assume that it remains correct. We have no data to indicate to which extent routers are moved without their names being changed, but it is possible that this occurs.

**Detectability**  It is impossible to detect someone’s intent in reading router names. Acquisition of the knowledge of which routers are on the route to the target host may be detectable though, depending on how this is done.
Effort Using router names to infer locations requires a massive job of deducing naming policies for different ASes.

GeoCluster

GeoCluster is a technique where routing information and the knowledge of the geographical location of a few hosts is used to determine the location of hosts within the same routes as the known hosts. Hosts within the same route form a geographical cluster, with the geographic location determined by the known hosts within that route.

Reliability How the location of the known hosts is gathered is crucial. In [10] user-submitted information from several large web sites is used. As described above, user submitted information can always be erroneous. This can be mitigated by using a larger number of known hosts, with the assumption that a majority of users supply correct information. Apart from this the reliability is as for using Routing Information.

Accuracy The accuracy is as good or better than when using only Routing Information. This depends on how many known hosts are used for each cluster. Routing information might not give the area where the route is actually deployed, while GeoCluster does, provided that the location of the known hosts is not erroneous.

Freshness The user-submitted information may be dated, and users might have moved without updating their location information. This might lead to GeoCluster determining the incorrect geographical location of a cluster. The routing information used is of course subject to the same limitations as when using only Routing Information.

Detectability The detectability is identical with that for Routing Information, given that the location of the known hosts is gathered in a non-detectable fashion.

Effort The real effort with GeoCluster lies in obtaining and keeping the locations of the known hosts current. This might not always be straightforward as personal privacy may be a concern. Also the quality of this information needs to be verified, possibly by developing algorithms for deciding what information to trust.
### 3.4.3 Measurement Based

All use of measurement based techniques have the drawback that at some point traffic is generated to the target host, increasing the likelihood of detection. For the tracing to work at all the target host must reply to this incoming traffic. Also a comprehensive set of landmarks is necessary.

**GeoPing**

**Reliability** GeoPing relies on the target host answering probe queries, and uses the delay values produced in its calculation. However, there is no guarantee that these delay values correctly represent the delay along the path between a probing landmark and the target. The target is free to delay its replies as it likes, thus skewing any attempts to compare the delay vectors of different landmarks. Such behaviour might be detected using techniques described in Section 3.6.3 but that requires an inordinately large number of probes from each landmark.

**Accuracy** As described in Section 3.3 different similarity models for calculating which landmark exhibits the closest matching delay pattern to the target affect the accuracy. No matter how well a given similarity model performs, the accuracy of GeoPing is limited by the set of landmarks employed. This remains true even if landmarks are distributed according to demographic densities and the tiered approach is used.

**Detectability** The higher accuracy one wants the more traffic one has to generate to the target host. This increases the chances of detection, either by the target host or by others monitoring the Internet for particular traffic patterns. Placing landmarks according to population density may reduce the numbers of landmarks necessary to achieve a given level of accuracy, and thus decrease the amount of traffic generated. To achieve usable accuracy over a large geographical area without flooding the target host a tiered approach seems essential. To reduce the amount of simultaneous traffic to the target host during the trace operation, it is possible to let the different landmarks perform their measurements at different times. Despite these optimizations the chance of detection is significantly higher than for approaches using public information sources.

**Freshness** If all landmarks simultaneously perform their measurements the result can be assumed to be as fresh as possible. If on the other hand one lets the different landmarks perform their measurements at different times, this will prolong
the entire operation, and more importantly the possibility of the landmarks encoun-
tering different network conditions increases. Compared to approaches based
on information sources the results are very fresh.

Using existing measurements between the landmarks are of course possible, this
would result in a shorter time to complete a trace operation, but the measurements
used would stretch over an even longer period of time than if doing the measure-
ments asynchronously from the different landmarks.

**Effort** For GeoPing to produce reliable and accurate results within a region an
extensive set of landmarks is necessary. This requires access to such a set of
landmarks. Knowledge of the networks these landmarks are connected by might
also help improve the results.

**CBG**

**Reliability** Due to CBG’s use of a bestline calculated from measurements between
all the landmarks it is less susceptible to manipulated delay values than GeoPing,
but this would still negatively affect the result.

**Accuracy** Theoretically CBG can provide the exact location of the target. In
practice a safety margin is necessary to not underestimate the distance from any
landmark. Underestimation leads to an incomplete intersection, and the calculation
fails, see Figure 3.4(b). Figure 3.4(a) shows how overestimating solves this, at the
cost of accuracy. Of course it is possible to miss even when overestimating, as
shown in Figure 3.4(c). In Section 5.4.4 we look into tuning the overestimation to
be as small as possible without incurring underestimation.

**Figure 3.4:** The possible outcomes of varying the safety margin in CBG [15]. $\tau$ is
the target of the location attempt.

As with GeoPing the number and placement of the landmarks are vital to the degree
of accuracy attainable.
Detectability  The effort is about the same as for GeoPing, except that for calculating an as accurate bestline as possible more probes between the landmarks would be necessary. This leads to a more easily detectable traffic pattern.

Freshness  The freshness is exactly the same as for GeoPing.

Effort  The effort is about the same as for GeoPing, but one also needs to know the precise distance between all the landmarks, and not only their approximate locations.

GeoBud

Reliability  Reliability should be the same as for CBG. Decreasing the over-estimation by a known size, the routers’ buffer delays, should not result in more cases of underestimation.

Accuracy  As noted in Section 3.3 GeoBud is capable of increasing the accuracy with about 30% compared to CBG.

Freshness  Apart from the time needed to measure buffer delays GeoBud has the same freshness characteristics as CBG and GeoPing. As with the measurements between landmarks, buffer delays could be measured in advance of actual tracing operations, at the cost of the values being slightly dated.

Detectability  To the target GeoBud is identical to CBG. The only difference is that a large amount of additional traffic is generated to measure the buffer delay of the routers used. If a pattern in this additional traffic could be identified detection would be easier.

Effort  The effort involved in measuring and keeping up to date the buffer delays of routers in addition to the effort necessary for running CBG makes GeoBud very expensive. With the number of routers involved it is questionable if the increased accuracy would make up for the added effort.

IDMaps

IDMaps and similar techniques do not try to infer geographical location, but rather network distance. These network distances could be used by measurement based geolocation algorithms as input instead of performing direct delay measurements.
**Reliability**  IDMaps suggest placing landmarks according to clusters of Address Prefixes\(^6\) (AP) and measure distances between them. Since it is based on the distance between clusters of APs its reliability is about the same as that of Routing Information, but with regard to delay not location.

**Accuracy**  The original IDMaps strives to attain an accuracy within a factor of 2 to direct delay measurements. The accuracy depends on the number of landmarks used, but will never approach that of direct measurements while at the same time being scalable. [19] achieves better accuracy than IDMaps, but still far from that of direct delay measurements.

**Detectability**  Determining distance in IDMaps does not involve target host(s) directly, and it is a continual service supposed to be a part of the permanent infrastructure of the Internet. Thus it is impossible for a target to know if the service is used as part of a geolocation attempt.

**Freshness**  An update frequency of days or at the best hours is suggested in [18]. Thus current network conditions will not be reflected. The maximum time between updates for producing relatively accurate results is estimated to 7 days in [25].

**Effort**  The requirement to have a network of landmarks, preferably such that every AP cluster is in the vicinity of a landmark, results in a relatively large set of landmarks. Also the resources required for storing the distances between AP clusters should not be underestimated. In [18] the number of landmarks used leads to every landmark needing to store a list of several hundred thousands entries. Finally determining the clustering of APs is not trivial, while still feasible. Later distance map techniques have much lower requirements, but depends on the cooperation of the hosts one wants to know the distance to, and are thus out of the question [22, 20].

### 3.5 Improvements to Current Techniques

As described in Section 3.4 all of the current techniques have some drawbacks. Here we suggest improvements to mitigate some of the effects of these shortcomings.

\(^6\)[18] defines an Address Prefix as "a consecutive address range of IP addresses within which all hosts with assigned addresses are equidistant (with some tolerance) to the rest of the Internet".
3.5.1 Combining Information Sources and Measurements

By first querying available information sources, a limited region to perform active measurements within can be defined. Fossen did this to some extent in [6] but the concept can be extended to use multiple sources that are checked against each other for correlation. This could increase confidence in the assumed region, or if the sources disagree, result in rejection of the assumption that the suggested region is correct. Better results could probably be obtained by weighting the information sources according to their relative scores in the categories discussed in Section 3.4.

3.5.2 Dynamic Regions and Super-Landmarks

Guye et al suggests in [14] to use GeoPing with a tiered approach as described in Section 3.3. We propose to do this in a more flexible way, by not operating with different static tiers, but by selecting landmarks dynamically. By using information source queries as a heuristic to narrow the assumed area of possible location a set of supposedly geographically close landmarks can be selected, as described above. Also based on the assumption in [13] that a host is most likely located in a densely populated area, a super-landmark can be selected. The criteria for choosing a super-landmark would be that the landmark has a central location within the assumed region, and/or is located in a densely populated area, or between multiple such areas of the region. The purpose of this super-landmark would be to confirm or invalidate the assumed area of location. The measured delay from the super-landmark would be compared to a threshold value based on the density of landmarks in the region and its size. By using a super-landmark for validation of the assumed region the amount of traffic to the target host could be significantly decreased, especially if the assumed region turned out to be incorrect. Another possibility would be to narrow the region even further by selecting only landmarks within twice the delay distance from the super-landmark to the target host, see Figure 3.5. Employing this technique with CBG would most likely yield better results than with GeoPing, as assumptions about correlation between delay and geographical distance would be necessary and are already part of CBG.

Alternatively GeoPing or CBG could be run with input from an IDMaps-like service to limit the region to perform active measurements within. This would limit the traffic to the target host. If this would result in a more accurate assumed region than the use of public information sources is dependent on the specific IDMap-like technique used, and the numbers of landmarks employed for delay measurements in this technique.

Note that depending on the delay value and the size of the assumed region this might actually increase the number of selected landmarks.

---

26
Figure 3.5: Selection of landmarks, shown as triangles, by the use of a super-landmark, shown here as a star. The dots symbolize population centers. The innermost circle is the delay from the super-landmark to the target, the outermost double this value. Landmarks inside the outermost circle are used to geolocate the target (not shown).

3.5.3 Limited Knowledge of Landmark Locations

Common to all measurement based techniques discussed so far is the assumption that the location of all landmarks are known by all the other landmarks. In a widely deployed system with many landmarks, maybe operated by different parties, this might not be desirable. As the number of landmarks increases so does the probability of one or more landmarks being compromised. If an adversary acquired the exact location of all landmarks in the system its efficiency could be severely lessened.

Another aspect is that the different parties might not want to disclose the location of their landmarks to other parties. Intuitively this might seem impossible, at least when using CBG. However, a mathematical technique called multi-party computation (MPC) can in fact compute the final result, without the parties divulging their locations to each other. MPC was introduced in [29] with later important contributions in [30, 31, 32].

Multi-Party Computation

MPC is in essence distributed computation performed by multiple parties, where the parties each hold information they do not want the other parties to know, but that is needed in the computation. Each piece of secret information is split into a number of shares, and one share distributed to each of the participating parties. The splitting must be done so that a single share does not divulge the content of the information. Each party performs the required calculation on the shares it has received and distributes the result. Recombination of the computed results from all parties constitutes the final result.
An important aspect of MPC is the lack of a trusted third party. Instead of placing their trust in an external party, or some specific subset of other participating parties, the parties trust that a majority of the participants are honest [33]. Thus for the locations to be released when using MPC, a number of the participating landmarks large enough to break this level of trust would need to cooperate in unveiling the locations of the rest, and in doing so also revealing their own locations to each other. Also, it is possible to detect incorrect computation by dishonest parties. Reistad has demonstrated that the theory of MPC can be used in a geolocation context. Although [36] limits itself to simple triangulation of points, it shows that there is no restriction in MPC that makes more advanced geolocation impossible.

Figure 3.6: CBG converted to use grid coverage for constraint representation.

CBG using MPC

Implementing CBG using MPC immediately presents a problem. The algorithm for computing the confidence region \( R \) requires knowledge of all landmark locations. See Section 4.2.1. To avoid this the geographical constraints can be expressed as boolean coverage within a grid reference system, instead of as functions of landmark location and distance. For each constraint the grid squares would be assigned the value 1 if included in the constraint, and 0 otherwise. Figure 3.6 shows the visual representation of a confidence region in the original CBG and converted to use boolean grid coverage. To achieve this the three constraint circles in the figure each have to be converted to boolean representation, split up and distributed to the participating parties. The function performed by all parties is to take the

---

8It has been shown that even if the majority is dishonest it is possible to keep the private information secret [34, 35]. This requires a gradual release of information by the parties, and leads to more complex computation.

9The best line used for constraint calculation at each landmark in CBG is not dependent on the landmark knowing the locations of the other landmarks, only its own distance to them. This lessens the secrecy of a landmark’s location somewhat, but does not defeat the secrecy achieved by MPC.
boolean intersections of all received pieces covering the same geographical areas. The final result, shown in Figure 3.6 as the region covered in 1’s is the boolean intersection of these distributed results.

Appendix D describes the Universal Transverse Mercator (UTM) grid reference system. For accurately representing CBG constraints a reference system with a more fine-grained grid than that of UTM is necessary. A NATO system called Military Grid Reference System which is based on UTM provides a precision down to 1 m, and could be used for this purpose [37].

Implementing GeoPing using CBG would require less adaption, at least as long as the Euclidean distance is used for determining the nearest neighbor, as MPC directly supports mathematical less than. However, as GeoPing uses the actual location of the landmarks as location estimation, the purpose of using MPC could be defeated by revealing the result.

**Figure 3.7:** A possible configuration of limited MPC. Measurement nodes A, B and C form a trust cluster with H as the TTP. MPC is used between the information exchange hubs inside the grey area.

**A Limited MPC Configuration for Geolocation**

A downside to using MPC is that the landmarks would need to exchange much larger amounts of information than without it, possibly resulting in a more detectable traffic pattern. Also with current MPC algorithms the amount of computation necessary grows rapidly when the computational complexity increases [32]. Applying MPC in a limited fashion, by introducing a tier using trusted third parties (TTP), may help to mitigate this. Figure 3.7 shows a possible configuration where the use of MPC is limited in this way. The measurement nodes denoted by A, B, C share an information exchange hub, H, and use it as their TTP, forming a trust cluster. A, B and C do not share any information between themselves, and trust H to not divulge any location
information about them to other parties. H achieves this by participating in MPC with other hubs, representing other trust clusters. Keeping the locations and information about the information exchange hubs secret is not important since they do not actively participate in the measurements themselves\textsuperscript{10}. Note that which measurement nodes are connected to which information exchange hubs are not dependent on geographical location or network topology but on trust. The number of information exchange hubs can be varied to balance the need between trust and performance. Fewer hubs would lessen the information exchange needed and thus increase performance, but more trust would be placed in each hub. The use of hubs has an additional advantage when using CBG. A hub combines any overlapping constraints of the measurement nodes in its trust cluster to a single constraint before participating in MPC. Thus identifying the location of the measurement nodes from the constraint shares becomes even harder.

The above tiered approach could also be used without MPC, to introduce a layer of some secrecy for the measurement nodes, but where all information exchange hubs would have to trust each other.

Employing MPC or other techniques for keeping the locations of the landmarks confidential would result in the techniques described in Sections 3.5.1 and 3.5.2 becoming less effective or outright impossible to implement.

### 3.6 Delay Measurement

Delay measurement is at the core of all of the active measurement techniques described in Sections 3.3 and 3.4. The quality of the measured delays have a significant impact on the results produced by the trace operations, especially in CBG, where delays are converted into actual geographical distances. GeoPing is not as much dependent on the correctness of measurements as on their consistency, as delays are compared against each other and not converted into real distances.

#### 3.6.1 Delay Components

The delay between two arbitrary hosts, A and B, in a best effort packet switched network can be expressed as in Equation 3.1.

\[
d = d_t + d_p + q + \varepsilon
\]

Transmission delay \(d_t\) is the time between the first and last bit of the probe has left A, and correspondingly arrived at B, see Figure 3.8. Propagation delay \(d_p\) is the\textsuperscript{10} Tracking measurement nodes by snooping traffic to known information exchange hubs would be possible.
physical minimum time necessary for the probe to travel from A to B. Queueing delay $q$ is the time spent in non-empty router and host queues. Random delay $\varepsilon$ is time wasted due to media access contention, router processing overhead, ARP\textsuperscript{11} resolution and other network disturbances. The combination of $d_t$ and $d_p$ is often referred to as deterministic delay, as they are constant along a link, while $q$ and $\varepsilon$ is known as stochastic or variable delay \cite{38}. $d_t$ is almost always negligible, due to small probe size and fast interfaces. Thus $d_p$ is what we really want to measure. In practice this is impossible to do accurately, due to unknown and varying size of $q$ and $\varepsilon$\textsuperscript{12}. Estimating the value of $\varepsilon$ and whether or not the probe is delayed due to queueing is therefore important \cite{39,40}.

![Figure 3.8: The different components that make up network delay.](image)

### 3.6.2 Ways to Measure Delay

Network delay can be measured in several ways, with different feasibility and certainty. Independently of the measurement technique, it is important to keep in mind that the Internet is a best effort\textsuperscript{13} packet switched network\textsuperscript{14}. This has important implications for delay measurements, as packets do not travel along a predefined circuit with given properties. The conceived properties of the path vary depending on, amongst other factors, traffic load and routing policies. What is perceived as the best path between two hosts may change at any time\textsuperscript{15}, due

---

\textsuperscript{11}Address Resolution Protocol. Used to find the MAC address from the IP address.
\textsuperscript{12}In [RFC2330], Framework for IP Performance Metrics, the term "wire speed" is used for the combination of $d_p$ and $\varepsilon$.
\textsuperscript{13}This might be about to change, as recent debate over net neutrality seems inclined towards service differentiation [net06].
\textsuperscript{14}Strictly speaking the Internet is not a single network, but a network of networks. Internally some of these networks may not employ packet switching, but traffic between the networks are packet switched, regardless of their internal workings.
\textsuperscript{15}In [41,42] more than 87% of paths were found to be stable over hours, and less than 2% experienced route changes more often than every 10 minutes. All examined paths remained stable for at least 60 seconds. Note that these numbers are from some years back, and might no longer be representative.
to downed links, congestion and changing routing policies, influencing the delay. Also, the path from B to A is often not the reverse of the path from A to B, as routing policies and other restrictions do not necessarily behave symmetrically [43, 44].

Independent of how the delay is measured the IP version used in the network may influence the results. An implication of the 128bit address in IPv6 is that each IP-packet becomes larger, and this results in bigger overhead, that translates into less efficient bandwidth usage, and higher latencies. Header compression can partly mitigate this, but in turn requires processing time for compression and decompression. [45] reports on the differences for RTT in an academic research network, and finds that IPv6/ICMPv6 RTTs generally are 0.4ms higher than IPv4/ICMPv4 RTTs for all packet sizes. Also immature and less optimized IPv6 stacks in routers may add additional extra delay in comparison to IPv4.

In Chapter 5 we compare IPv4 RTT and one-way delay, and IPv4 and IPv6 RTT in the Uninett network.

**Round Trip Time**

Round Trip Time (RTT) is widely used due to its simplicity [46][RFC1305]. It is actually a double delay, made up from the delay from A to B, and from B to A. These two parts do not necessarily contribute equal shares to the total RTT, for reasons discussed above.

The term probe used above is an abstraction for the actual packets traversing the network path. What constitutes a probe depends on the protocol and technique used. Tools like ping and traceroute send an ICMP_ECHO packet from host A and waits for an ICMP_ECHO_REPLY packet from host B. The default size of each of these packages is 64 bytes and together an ECHO and REPLY pair constitute an ICMP probe.

Some Internet service providers and host operators filter, drop or down-prioritize ICMP packages [47][pin06]. This results in ICMP-based tools not being able to reach all hosts, and reported delays may be larger than necessary. To get around the limitations imposed on ICMP traffic some tools employ probes based on TCP [1]. These tools use a technique called TCP-ping, where host A tries to establish a TCP-connection with host B by sending a TCP SYN packet. Host B replies with a TCP SYN-ACK or RST packet.

---

16 Techniques for header compression without incurring processing delays exist, but are not in common use [45]
17 Some implementations of traceroute use UDP.
18 Without any extra options.
19 It is difficult to give a good estimate of how large a proportion of Internet hosts are unreachable by ICMP. [39] reports that more than 12% of about 20,000 probed hosts were unreachable.
The TCP-ping solution has its own drawbacks. TCP traffic must be directed not only at a host, but at a specific port number. There is no universal TCP port number that all hosts are required to listen on. A common way around this is to use port 80 (http) or some other commonly used port, that many hosts are assumed to listen on. When receiving a TCP SYN packet most hosts do not reply right away, but try to match or spawn a process to handle it, incurring extra delay. ICMP packets are in contrast replied to immediately.

Most networks prioritize TCP-traffic, thus queueing delay is minimized. Additionally TCP SYN packets are 40 bytes, resulting in a marginally lower transmission delay than for ICMP packets. In practice the RTT measured using ICMP and TCP probes are often almost identical, with a correlation above 0.99 for the over 100 sites measured in [39].

In our prototype application in Chapter 4 we use RTT, and take into account the problems described above by implementing the techniques suggested in [39, 40].

**One-Way Delay**

With access to synchronized time at both host A and B it is possible to measure one-way delay. This is done by time stamping a probe consisting of a TCP packet when it is sent from A, and subtracting the value of this time stamp from the current time when the probe arrives at B. Synchronized time is usually achieved by using the Global Positioning System (GPS) as reference and synchronization source. The requirements for measuring one-way delay makes it impossible to use in many situations, but it is an interesting metric for checking how good an estimation halved RTT is for measuring delay.

Recently Gurewitz et al. have come up with a novel approach for estimating one-way delay not requiring synchronized time [38]. The approach consists of identifying as many independent cyclic paths between hosts as possible, and performing one-way measurements in both directions along these paths. The paths need not be symmetric. The theory is that along a cyclic path clock offsets are canceled out, and the total one-way delays along all cyclic paths can then be used as constraints for estimating the actual one-way delay, using an objective function. The results achieved outperform halved RTT for the paths examined, but do not quite match GPS assisted measurements. The technique is possible to implement in almost any IP based network, as no non-standard protocols are used. Other alternatives to GPS synchronized time are presented in [48, 49] but these require symmetric paths to function accurately.
3.6.3 Confidence in Delay Measurements

Independent of the approach used to measure delay, we would like to be able to say something about the confidence of the results, and if possible estimate the $\varepsilon$ part discussed in Section 3.6.1. The confidence of a delay measurement result can be thought of as the probability of the delays observed being representative for an uncongested path, that is a path where the queuing delay $q$ in Equation 3.1 is zero.

Confidence Regions and Detecting Congestion

Congestion can substantially affect the results of delay measurements. Therefore it is important to identify the occurrence of congestion(s) during measurement runs. Zeitoun et al devised a way to determine confidence and $\varepsilon$ for RTTs by comparing the RTT values of probe pairs to detect congestion [39, 40]. A probe pair is defined as $RTT_n$ and $RTT_{n+1}$ where $n$ is the probe’s sequence in the sample. Three congestion regions are defined:

- Region C1: Both probes see empty queues and experience minimum RTT plus $\varepsilon$.
- Region C3: Both probes always see a queue, and thus persistent congestion.
- Region C2: One of the probes experience queueing delay but the other does not. This indicates a transient congestion.

To determine the congestion regions minimum RTT and $\varepsilon$ are used. The minimum RTT determines the bottom left corner of Region C1, see Figure 3.9. $\varepsilon$ determines the boundaries for C1, C2 and C3. On a congestion-free path most probes should be within Region C1.

The mode RTT is very close to the minimum RTT in a normally distributed RTT sample, and a large number of RTTs are within 10% of this mode [50]. This makes it possible to calculate $\varepsilon$ using the observed minimum RTT and the mode RTT. $\varepsilon$ is equal to the size of the window around the most frequent values of RTT, or double the difference between the minimum and mode RTT.

A point $p_i$ in the phase plot represents a probe pair. The point $p_i = (RTT_i, RTT_{i+1})$ is part of C1 if $RTT_i \leq (\min RTT + \varepsilon)$ and $RTT_{i+1} \leq (\min RTT + \varepsilon)$. It is part of C2 if $\max(RTT_i, RTT_{i+1}) > (\min RTT + \varepsilon)$ and $\min(RTT_i, RTT_{i+1}) \leq (\min RTT + \varepsilon)$. And finally it is part of C3 if $RTT_i > (\min RTT + \varepsilon)$ and $RTT_{i+1} > (\min RTT + \varepsilon)$.

\[\text{The most frequent values in a data sample is known as the mode of the sample [sta06].}\]
Figure 3.9: A phase plot of a RTT sample showing the distribution among the three congestion regions [40]. \( \tau \) denotes the inter-probe delay.

Computing the Confidence

The confidence is expressed as \( C_1 + C_2 + C_3 = 1 \). With \( N \) probe pairs the value of \( C_1 \) is computed as in Equation 3.2,

\[
C_1 = \frac{1}{N} \sum_{p_i}^{N} \frac{1}{\Delta(RTT_i)} \times \frac{1}{\Delta(RTT_{i+1})}, \text{ where } \Delta(x) = \begin{cases} 
1 & x = \min RTT \\
\left\lceil \frac{x - \min RTT}{\epsilon} \right\rceil & x > \min RTT
\end{cases}
\]

(3.2)

\( C_2 \) is computed as in Equation 3.3.

\[
C_2 = \frac{1}{N} \left[ K - \sum_{p_i \in C_2} \frac{1}{\Delta(max(RTT_i, RTT_{i+1}))} \right], \quad (3.3)
\]

where \( K \) is the number of probepairs in \( C_2 \).

And \( C_3 \) is computed as in Equation 3.4.

\[
C_3 = \frac{1}{N} \left[ M - \sum_{p_i \in C_3} \frac{1}{\Delta(max(RTT_i, RTT_{i+1}))} \times \frac{1}{\Delta(RTT_{i+1})} \right], \quad (3.4)
\]

where \( M \) is the number of probepairs in \( C_3 \).
Equations 3.2 to 3.4 weigh the points such that the closer a point is to C1 the more important it is to the final value. The $\Delta(x)$ function in Equation 3.2 is used to calculate the distance in regions between minimum RTT and the given RTT in all the equations.

The implementation by Wang et al uses TCP probes and is written in C. We have re-implemented it in Java, using ICMP probes, see Section 4.3.3 and Appendix C.5. In Chapter 5 we perform several tests measuring the C-values and $\varepsilon$ to analyze the performance of the Uninett network and the effect on geolocation of varying probe parameters.

### 3.6.4 From Delay Measurements to Geographical Distance

The speed of light in optical fibers is approximately $1.962 \times 10^8$ m/s [51]. This is the basis for conversion of delay measurements to geographical distance. However, this conversion is not straightforward. At this speed, 1ms translates into 196.2 km, making accurate delay measurement paramount. But even if spot-on delay measurements were possible there are other important sources of error. Cables are not laid out as the crow flies, they meander through the landscape, following roads, rail tracks, elevations and other topological properties. Thus cable distance is always longer than actual distance, also known as great circle distance, see Appendix D.3. In most cases, knowing the physical topology of more than small parts of the Internet in detail is practically impossible, making the calculation of the offset between geographical and cable distance an educated guess at best.

To complicate matters further [52, 53, 54] show that current BGP inter-AS routing policies tend to exhibit path inflation, making the discrepancy between the path travelled by packets and geographical distance even larger. Although the degree of path inflation seems stable over time, it differs between long and short paths, and between different size Internet Service Providers. Accurate numbers for how large a portion of all paths exhibit inflation, and by how much are not agreed upon, due to different data sets and methodologies. [53] suggests that as much as 80% of all paths are inflated, and that 20% are inflated by at least 50%. On the other hand [52] suggests that about 45% of all paths are inflated.

In Section 5.1.1 we look at the difference between minimum theoretical delay as a function of great circle distance and actual measured minimum delay.

### 3.7 Internet Protocol v6

Internet Protocol v6 (IPv6) is the next generation Internet Protocol [RFC2460]. IPv6 is a conservative extension of IPv4, but differs from it in several aspects. We will only touch upon differences relevant to geolocation.
3.7.1 Address Space and Assignment

IPv6 extends the address space from today’s 32 bit to 128 bit, resulting in an immense increase in the number of unique addresses. This in itself is not very interesting from the point of view of geolocation. However, as a result of the massive address space, IPv6 also calculates and distributes IP-addresses differently from its predecessor. This has important implications for the traceability of addresses.

In IPv4 addresses are either statically assigned or distributed by DHCP servers. In addition to these methods IPv6 introduces stateless auto-configuration [RFC2462], where hosts generate their own addresses based on a combination of two logical parts; a (sub-)network prefix and a locally generated host part. The host part is most often derived from the globally unique MAC address, and offers an opportunity to track user equipment, and so users, across time and address changes. This loss of anonymity has been addressed in [RFC3041][55], by different schemes for host part randomization. As a more extreme measure to preserve anonymity it has been suggested to use a new IP address for every TCP connection [56].

Since IPv6 addresses are plentiful, it is reasonable to allocate addresses in larger blocks than for IPv4, which makes administration easier and avoids fragmentation of the address space. This in turn leads to smaller routing tables, and more efficient routing. A less fragmented address space might make techniques like IDMaps and GeoCluster more accurate.

3.7.2 Mobile and Hierarchical Mobile IP

Mobile IP (MIP), an optional extension to IPv4 [RFC3344], is an integrated part of IPv6 [RFC3775, RFC3776]. With mobility being an integral part of the protocol, and more and more IP capable mobile devices an explosion in the number of mobile nodes (MN) is expected. Also 3GPP2, one of the two consortiums publishing competing third generation mobile phone standards, has decided to build their standard on MIPv6.

The goal of MIP is to let a MN keep the same IP address wherever it is. MIPv4 uses two IP addresses per MN to achieve this; a home address and a care-of address (CoA). The home address is static and used to identify the MN, while

---

21Unique addresses in IPv4: 4,294,967,296.
Unique addresses in IPv6: 340,282,366,920,938,463,463,374,607,431,768,211,456
22Dynamic Host Control Protocol. Defined in [RFC2131]
23Media Access Control. (Globally) unique equipment identifiers used in many communication networks for identification at layer 2 in the OSI network stack.
24In current policies an end-user is allocated 64 bits of IPv6 address space, while organizations are allocated 96 bits or more.
the CoA changes with each change of network attachment. To work MIPv4 requires two additional network nodes; a Home Agent (HA) and Foreign Agent (FA). Whenever the CoA changes this is registered in the HA. The task of the HA is to relay incoming traffic to the current CoA. The FA is responsible for allocating an IP address and related configuration information to the MN at its current location. The MN may be configured to route return traffic through its HA or directly to any Correspondent Node (CN). If the traffic is routed through the HA it is impossible for outsiders to know the location of the MN, or its CoA without snooping the packets sent between the HA and MN. Figure 3.10 shows the normal case where a CN sends a request Req to the MN’s home address. This request is forwarded as T req by the HA to the MN by use of tunnelling. The reply Rep is sent directly from the MN’s CoA to the CN.

![Diagram of Mobile IP](image)

**Figure 3.10:** Triangle routing in Mobile IP.

Due to the auto-configuration of host addresses described in Section 3.7.1 MIPv6 has no FAs. In MIPv6 it is also possible to avoid the triangle routing described above, by the use of binding updates. Binding updates lets the CN in Figure 3.10 send subsequent requests directly to the MN’s CoA. This leads to increased performance and is the default behaviour. For geolocation purposes this is an advantage as the MN’s CoA is public. Note that this could still be negated by the use of one-time CoAs as described in Section 3.7.1.

An extension of Mobile IPv6 known as Hierarchical Mobile IP (HMIP) [RFC4140] has been proposed to lessen the number of updates from MNs to HAs. HMIP partitions the Internet into different administrative domains, and allows MNs to roam freely inside a domain without updating its HA. This is accomplished by the use of a network node named Mobility Anchor Point (MAP). The MAP acts as a local HA to the MN within the domain and assigns it a publicly visible Regional care-of address (RCoA). The MN also has a Link care-of address (LCoA) that is used between the MN and the MAP. The MN can choose to not divulge its LCoA to CNs and its HA. If HMIP is used, and the MN does choose to hide its LCoA, it is not possible to determine the location of the MN more accurately than that it is inside the HMIP region.

---

25 A domain might be partitioned further with more MAPs at different levels within the domain.
In this chapter we give a short introduction to MAFIF, the Multi-Agent Framework for Internet Forensics, we developed in [1] and related technologies. The main part of the chapter is dedicated to describing the design and implementation of geolocation functionality in this framework.

4.1 The Existing MAFIF Framework

MAFIF is based on JADE, a framework for developing multi-agent applications in Java [57]. A condensed presentation of MAFIF and some results of the content securing application built on it is available in the form of an article draft in Appendix A. The full design and implementation of the framework is available in [1], which also discusses the advantages and drawbacks of using multi-agent technology, as well as security implications.

![Figure 4.1: A High-Level Design of MAFIF, showing the different agents.](image)

39
A high-level design of the framework, and its agents is shown in Figure 4.1. Different applications are developed as sets of agent behaviours. A behaviour is a piece of functionality that an agent uses to execute a certain task. Some behaviours make up the basic functionality of the different agents. Communication among the agents is based on message passing, and the reception of messages and dispatching to appropriate behaviours is handled by a Receive behaviour in each agent.

JADE defines an environment called a container. A platform consists of one or more containers, running on one or more hosts. MAFIF provides five distinct types of agents to handle different tasks. For each container there are single, non-transient AdminAgents, LogAgents and TimeAgents, shown as lettered triangles in Figure 4.1. Additionally transient SessionAgents and WorkerAgents are created as part of the execution of applications. The non-transient agents handle application independent tasks like time-keeping, logging and starting the execution of applications. Depending on the application SessionAgents and WorkerAgents are created with different behaviours.

All communication between agents on different containers is encrypted, and all communication, whether intra- or inter-container, is signed.

### 4.1.1 Command and Work Flow

The AdminAgent acts as a coordinator, and upon receiving a request from an operator for some investigation action creates a transient SessionAgent. Based on the type of investigative action the SessionAgent takes the necessary preliminary steps to decide upon the number and type of WorkerAgents it needs to create, how to distribute them, and how to share the load among them. Upon creation the WorkerAgents immediately start their assigned work. When all WorkerAgents of a session have terminated, the SessionAgent informs its AdminAgent before it too terminates.

The above program flow is an optimization with regard to the original MAFIF framework, where a WorkerAgent upon creation would query its SessionAgent for a work assignment. The SessionAgent would reply with a suitable assignment, and first then would the WorkerAgent start on its real task. This resulted in two extra messages being exchanged per WorkerAgent. Informal experiments show that this greatly improves the speed of the content securing application, and probably the scalability of MAFIF as well.

Figure 4.2 shows an Agent UML sequence diagram for the execution of a geolocation attempt, including most communication between the different agents. JADE uses ontologies for defining the content of agent messages [58]. The ontology used in MAFIF is defined by us, and is described in great detail in [1].

---

1The creation and distribution of work to WorkerAgents and the amount of messages exchanged in this process is probably the part of MAFIF most likely to act as a bottleneck.
4.1.2 Agent UML (AUML)

The sequence diagram in Figure 4.2 is in a notation named AUML, an extension of UML adapted for modelling agents. AUML was pioneered by Bauer in [59]. We use a custom notation based on a combination of Bauer’s notation in [60] and that of Huget in [61]. Our notation and the reasoning behind it is explained in [1]. In short it is a compromise between power of expression and readability. The class diagrams of agents and behaviours in Appendix B is in this notation.

4.2 Geolocation Algorithms

We have implemented CBG and GeoPing as a set of agent behaviours. The relation among the different agents and their behaviours that constitute these two techniques is shown in Figures 4.2 and 4.3. The preliminary steps of the two algorithms have much in common, and to avoid code duplication we have merged these steps. This also simplifies the logic and reduces the number of behaviours, although at the cost of not having a distinct set of behaviours for each of the algorithms. The functionality for performing delay measurements that form the basis of both the algorithms are explained in Section 4.3.
4.2.1 CBG

The bestline used by CBG, described in Section 3.3, is the line that captures the least distorted relationship between geographic distance and network delay from each landmark to all other landmarks. Figure 4.4 shows a scatter plot with the bestline and baseline of a landmark. The baseline represents the theoretical lowest delay at any point, its slope \( m = 1/100 \) defined by the propagation speed of light in optical fibers, see Section 3.6.4. We investigate the relation between the lower delay limit and actually observed delay in the Uninett network in Section 5.1.2.

\[
y - d_{ij} - b_i x - b_i \geq 0, \forall j \neq i
\]

(4.1)

The bestline for each landmark can be defined as the line \( y = m_i x + b_i \) that is closest to, but below all points in the plot. Since negative delays are impossible,
the line’s intercept must be non-negative. This can be expressed as in Equation 4.1, where \(d_{ij}\) is the delay from landmark \(L_i\) to each landmark \(L_j\), with \(i \neq j\) and where \(g_{ij}\) is the corresponding geographical distance. The slope \(m_i\) is defined as in Equation 4.2.

\[
m_i = \frac{d_{ij} - b_i}{g_{ij}}
\]  

(4.2)

To find the values for \(b_i\) and \(m_i\) from Equation 4.1 Equation 4.3 is used. We have implemented this in the behaviour `WorkerTraceCalculate` by finding the point with the lowest delay, and solving Equation 4.1 with \(b_i\) and \(m_i\) as unknowns for this point and every other point with a larger \(d_{ij}\). The bestline is defined as the lowest \(m_i > m\) with a corresponding \(b_i \geq 0\). The source code of `WorkerTraceCalculate` is available in Appendix C.3.3.

\[
\min_{b_i \geq 0, m_i \geq m} \left( \sum_{i \neq j} y - \frac{d_{ij} - b_i}{g_{ij}} x - b_i \right)
\]  

(4.3)

The geographical constraint \(\hat{g}_{i\tau}\) from each landmark to the target \(\tau\) is calculated by Equation 4.4. This constraint actually represents a circle \(C_{i\tau}\) with the landmark at its center and \(\hat{g}_{i\tau}\) as its radius. A single landmark has no notion of the direction to the target, only of the overestimated distance to it.

\[
\hat{g}_{i\tau} = \frac{d_{i\tau} - b_i}{m_i}
\]  

(4.4)

The final step in CBG is to take the intersection of all the \(C_{i\tau}\)’s defined by the geographical constraints and radii of the landmarks, and compute its area and
center. The region $\mathcal{R}$ intersected by all $C_{i\tau}$ is the confidence region of the target $\tau$'s location. This intersection is defined as in Equation 4.5, where $K$ is the total number of landmarks.

\[
\mathcal{R} = \bigcap_{i=1}^{K} C_{i\tau} \tag{4.5}
\]

Performing Equation 4.5 requires one to find all points of intersection between the $C_{i\tau}$'s and compute the area of the convex hull\(^2\) of these points. To determine the convex hull of a set of points the points must be sorted into a list that only includes the points representing the vertices of the convex hull in a correct winding order. Fortunately all the intersection points between $C_{i\tau}$'s are vertices in the convex hull in the case of CBG, as $\mathcal{R}$ is by definition convex \(^{15}\).

In a $xy$-defined two-dimensional plane it is relatively simple to find all intersection points. However, $\mathcal{R}$ is not defined in a two-dimensional plane but on the surface of the Earth, which is spherical\(^3\). To correctly calculate the intersection points defining $\mathcal{R}$ on a sphere we use functionality that is part of OpenMap, see Section 4.5.3. OpenMap returns the intersection points in correct order defining a convex hull. Although OpenMap is capable of conversion of point coordinates between the two-dimensional plane represented by the screen and the latitudes and longitudes of the Earth it does not do area calculations. The area of $\mathcal{R}$ is computed using Equation 4.6, which is a general formula for computing the area of a spherical polygon of arbitrary shape with edges defined by great-circle arcs \(^{62}\). $n$ is the number of vertices in the polygon, $\sum \alpha_i$ is the sum of all the interior angles between all the vertices and $R$ is the radius of the sphere.

\[
A_p = R^2 \left[ \sum \alpha_i - (n - 2)\pi \right] \tag{4.6}
\]

To determine the interior angles required in Equation 4.6 it is necessary to divide the spherical polygon into a set of spherical triangles, see Figure 4.5(a). The calculation of the interior angles is then done using Equation 4.7 for every corner in every spherical triangle. Each of the edges $a$, $b$ and $c$ of the spherical triangles is defined as the angle between the pair of points on the sphere defining the edge, see Figure 4.5(b).

\[
\cos(a) = \cos(b) \cos(c) + \sin(b) \sin(c) \cos(A) \tag{4.7}
\]

\(^{2}\)A convex hull is the smallest possible convex polygon circumscribing all the points in a set $P$ of points.

\(^{3}\)The Earth is not a perfect sphere, see Appendix D for more on this.
\( \mathcal{R} \) might contain edges more curved between any two vertices than the corresponding great circle arc between the same vertices. Thus the result of Equation 4.6 will be a slight underestimation, similar to that of Figure 4.6.

**Figure 4.6:** The difference between the area of a convex hull approximation, shown as the dark grey area, and the real area of an intersection of circles, shown as the combination of the dark and light gray areas combined.

To calculate the centroid of \( \mathcal{R} \) we do an approximation by calculating the centroid of the two-dimensional representation of \( \mathcal{R} \), and using OpenMap, convert the resulting \( xy \) coordinates into latitude and longitude. The area of a planar convex polygon is calculated using Equation 4.8, where \( x_n, y_n \) defines the nth vertex. The coordinates of the polygon’s centroid is calculated using Equations 4.9 and 4.10.

\[
A_{\mathcal{R}} = \frac{1}{2} \sum_{n=0}^{N-1} \begin{vmatrix} x_n & x_{n+1} \\ y_n & Y_{n+1} \end{vmatrix}
\]

\[ (4.8) \]

\[
c_x = \frac{1}{6A} \sum_{n=0}^{N-1} (x_n + x_{n+1}) \begin{vmatrix} x_n & x_{n+1} \\ y_n & Y_{n+1} \end{vmatrix}
\]

\[ (4.9) \]

\[
c_y = \frac{1}{6A} \sum_{n=0}^{N-1} (y_n + y_{n+1}) \begin{vmatrix} x_n & x_{n+1} \\ y_n & Y_{n+1} \end{vmatrix}
\]

\[ (4.10) \]
The final step of calculating the area of $\mathcal{R}$ and its centroid is not implemented as part of the multi-agent framework, but as stand-alone classes included in the GUI, since GIS functionality is required to correctly compute distances between geographic locations.

Currently the intersection algorithm does not allow for discarding $C_{i\tau}$’s that lead to an empty intersection at all. It should be possible to discard up to a certain threshold of $C_{i\tau}$’s if the remaining $C_{i\tau}$ strongly agree on a common intersection.

### 4.2.2 GeoPing

The implementation of the GeoPing algorithm creates a delay vector $DV'$ for the target and one delay vector $DV$ for each landmark. When all measurements are done the nearest neighbor is found, as explained in Section 3.3. Currently only Euclidean distance is implemented for finding the $DV$ that most resembles $DV'$. Equation 4.11 shows how this is done.

$$\Delta(DV) = \sqrt{(d_1 - d'_1)^2 + \ldots + (d_N - d'_N)^2}$$ (4.11)

The best matching landmark is the one with the smallest $\Delta(DV)$. The geographical location of the three best matching landmarks and their Euclidean values are returned for display to the user.

### 4.3 Delay Measurements

As described in Section 3.6.4 it is crucial to get as correct delay measurements as possible. Unfortunately we did not have access to passive measurement equipment, so we are forced to use RTT as an approximation. We perform our RTT measurements with ICMP-based ping.

#### 4.3.1 Use of Native Ping Binary

To get as little overhead as possible the native ping utility of the underlying operating system is used. It is reasonable to assume that this utility is much more optimized than any implementation we could come up with in the limited time scope of this project. Additionally Java’s support for ICMP leaves much to be desired, in fact third party libraries are necessary to get this functionality [jpc06, roc06]. These libraries depend on a mechanism called raw sockets. Not all operating systems support raw sockets, and those that do usually requires administrative privileges to access the functionality. Using ICMP directly from within Java is therefore out of the question.

46
Most operating systems provide a utility named `ping` to measure RTT between hosts using ICMP. Java provides a mechanism for starting external programs and receiving their output. Executing external programs and reading their output presents some challenges, as any input parameters must be hard-coded to some extent. The output must be parsed, and any change in output format may result in unexpected results. Currently our ping-wrapper supports the iputils version of ping used in most Linux distributions [ipu06].

Two sample outputs from ping can be seen in Figure 4.7. In Figure 4.7(b) no filtering is employed, and all packets arrive successfully. As can be seen in Figure 4.7(a), ICMP filtering is employed on the path, and some packets are lost. In this case the ones with ICMP sequence number 4 and 6 are missing, see lines 3-6. Variations where not all packets are filtered, all packets are dropped, or some packets are lost even if filtering is not employed, are of course possible. Our wrapper takes care of this by maintaining its own queue of ping items, and automatically inserts any lost or filtered packets. If it is not possible to measure a RTT to the host in question, within the number of pings specified, a `HostUnreachableException` is thrown. The case in Figure 4.7(a) would result in such an exception being thrown.

### 4.3.2 Result Confidence

Our wrapping of the native ping binary is quite flexible, and allows for specifying the number of pings to send and the distance between them. This is used to implement the confidence estimation techniques described in Section 3.6.3. Algorithms for calculating the C1 confidence and $\varepsilon$ values have been adapted from RTT-Ometer in [40]. The number of probes needed to accurately measure C1 and $\varepsilon$ with
reasonable confidence is too high for use in probing of targets, but is used to get as accurate measurements between landmarks as possible.

4.3.3 Relation of the Measurement Parts

Figure 4.8 shows the relation of the different Java classes that are directly concerned with RTT measurements. RTT measurements are performed by SessionAgents and WorkerAgents as part of their behaviours for trace operations. (A)UML diagrams are provided in Appendix B, source code in Appendix C.

![Figure 4.8: Relations of the Java classes concerned with RTT measurements.](image)

MinRTT, ModeNode and PingItem

MinRTT is the class wrapping the native ping binary. It also contains methods for keeping track of its queue of PingItems. Some of its calculation methods use the helper class ModeNode. This class calculates the statistical mode given an input array of floating point numbers. MinRTT is responsible for calculating the different C-values and $\varepsilon$.

PingBehave and WorkerTraceTarget

PingBehave extends JADE’s OneShotBehaviour and is simply a way for an Agent to use MinRTT with different parameters. WorkerTraceTarget extends
PingBehave. This is done so that it can update the Traced table with information about the newly traced target and call WorkerTraceFinal before terminating.

**WorkerTraceLandmarks**

WorkerTraceLandmarks extends JADE’s ParallelBehaviour and is used by WorkerAgents to execute several PingBehaves simultaneously. Before starting measurements to all the landmarks it checks to see how old the most recent measurements are. If they are found to be fresh enough the current values are used instead of performing new measurements.

### 4.4 User-Interaction and Management

The MAFIF prototype described in Appendix A left user-interaction and management largely as an open issue to be resolved later. There are several ways to connect a JADE platform to external applications, from integrating JADE in Java application servers like JBoss using JadeMX, to full-blown web service integration gateways [jad06] [63, 64]. These approaches offer a lot of functionality we do not need and also add a lot of complexity. JADE provides a simpler approach through the jade.wrapper.gateway classes. By using the functionality of these classes in a Java servlet access to the multi-agent system can be provided through an ordinary web page. Figure 4.9 shows how the different parts of our system are connected through the use of this gateway.

![Diagram showing information flow between the agent system and external parts. The gateway agent, denoted by a G, acts as a single point of entry into and out of the agent system.](image)

**Figure 4.9:** Information flow between the agent system and external parts. The gateway agent, denoted by a G, acts as a single point of entry into and out of the agent system.
### 4.4.1 Graphical User Interface

The provided GUI is very simple. It consists of a single web page, and input fields for specifying the address to trace and options like the algorithm and probe parameters to use. The information submitted by the user is read by a set of servlets that communicates with the agent system using the `jade.wrapper` gateway classes. When the agent system has finished the trace operation the servlets use GIS functionality to do the calculations described in Section 4.2.1 and create a visual representation of the trace result that is displayed to the user. A simplified program flow can be seen in Figure 4.10. In step 1 the user requests a trace operation through the web page, this is relayed to the gateway agent by the servlets in step 2. The gateway agent contacts an AdminAgent in step 3, this agent administers the execution of the trace operation internal to the multi-agent system. This execution is not shown in the figure, but described in detail in Sections 4.1.1 and 4.2. In step 4 the trace results are returned from the AdminAgent to the gateway agent, which in turn returns them to the servlets in step 5. The image constructed by the servlets is finally shown on the web page to the user in step 6.

![Figure 4.10: The sequence of events following a user-request to trace an address.](image)

The set of servlets and the relations between them is shown in Figure 4.11. `GetTrace` simply takes input from the user and forwards it to `GetMap`. `GetMap` takes as input an IP-address and optionally a cache ID. If a cache ID is submitted the image of an existing trace corresponding to the cache ID is retrieved in the form of a `TraceCacheEntry` from `TraceCache`. If no cache ID is submitted a fresh trace is performed if the address has not been traced within the last 24 hours. An entirely new trace can be forced before the 24 hour limit. `GetMap` shows a simple navigation menu to the user that makes it possible to pan and zoom the map image showing the trace result. The new image generated as a result of panning and zooming is generated by `GetMapImage` which takes as input an IP address and a cache ID and shows the corresponding map image.

The class `TraceCache` is only used internally by `GetMap` and `GetMapImage` to actually perform traces by interfacing with the multi-agent-system and query about cached traces. For each request `TraceCache` deletes any `TraceCacheEntry` inst-
ances older than 24 hours and returns a new TraceCacheEntry to the requester.

### 4.4.2 Management and Properties Files

Management of the agent system is currently done through properties files and scripting. JADE uses properties files for defining which services are to be part of a platform, and to specify host addresses, usernames and passwords and other configuration information. The JADE properties files and the management scripts are available in Appendices C.9 and C.8 respectively. The following scripts are provided:

- unidist for distribution of new versions
- unirun for starting the system
- unistop for stopping the system gracefully
- unikill for forcing the system to stop immediately
- unicdb for database creation

### 4.5 Geographical Functionality

As described in Section 3.6.4 conversion of delay measurements into geographical information is a key component of geolocation. To facilitate this conversion, and allow the trace data to be displayed in a meaningful way, we need functionality that can take the geographical properties of the Earth into account. A more in-depth discussion of the geographical properties of the Earth and mappings of it is provided in Appendix D.
Systems with this functionality is commonly referred to as Geographical Information Systems (GIS). There exist many toolkits and platforms for developing GIS. We do not have time for a complete comparison of all the alternatives, a limited comparison based on some key requirements is performed below. Developing this functionality from scratch is not an option, as this would be too time-consuming.

4.5.1 Requirements for GIS Toolkits

Below we give a list of requirements for GIS toolkits. These requirements are based both on the functionality needed in our application, and on non-functional aspects related to the development process.

- **Programming Language**
  Toolkits not employing Java will not be considered. Our existing framework is based on Java and introducing extra complexity by adding a different programming language is out of the question. By using the same language throughout the entire application better integration is also possible.

- **Standards Compliant**
  The Open Geospatial Consortium (OGC) [ope06a] publishes publicly available standards and specifications for geospatial and location based services. The goal of OGC is "to make complex spatial information and services accessible and useful with all kinds of applications".

  Our limited comparison might lead us to choose a less than ideal platform, and maintaining compliance with OGC standards will make a potential switch of toolkit at a later time easier.

- **Freely Available**
  The toolkit should be freely available, preferably as open source. This way we do not need to worry about any license-restrictions if we decide to deploy applications based on the framework on many hosts at a later time. Preferably the source should also be available, as we may need to modify the toolkit’s functionality.

- **Maturity**
  The range of functionality needed in GIS applications is very broad. Toolkits may focus on limited sets of functionality, thus providing unsatisfactory functionality overall. We need a stable, well-tested, and actively maintained toolkit with a broad and stable feature set.

- **Ease of Development**
  Due to the timescale of this project the toolkit needs to be easy to start using.

- **Support/documentation**
  Good support and documentation is absolutely necessary.
4.5.2 Comparison of GIS Toolkits

The toolkits below were evaluated for the purpose of geographical calculations and visualization, based on the requirements in Section 4.5.1. A comprehensive list of available toolkits (and other GIS software) not evaluated here is given in [ope06b]. Based on this evaluation OpenMap was chosen for implementing the needed GIS functionality in our application.

**Jump Topology Suite (JTS) and OpenJump**

JTS is developed by Vivid Solutions, and is published under the LGPL [jts06b, jts06a]. Its current version is 1.7, released January 19 this year. JTS in itself is an API providing a spatial object model and fundamental geometric functions. It implements the geometry model defined in the OGC Simple Features Specification for SQL [65]. JTS is written entirely in Java. JTS formed the basis of a program called JUMP Unified Mapping Platform (JUMP) developed by Vivid Solutions, now taken over by the open source community and renamed OpenJump. The documentation of OpenJump is rather scarce, and partially in German.

**Landserf**

LandSerf is developed by Jo Wood at the City University in London. It is not open source, but it is freely available and has a documented API to allow programmers to customize and enhance the software. Landserf is written in Java and the current release is 2.2 [lan06].

**Google Maps**

Google maps [goo06a] is not really a toolkit, but provides geographical functionality. It is based on Google AJAXSLT [aja06] which is heavily dependent on JavaScript and thus must be run in a web browser. AJAXSLT is provided under the BSD license, but Google maps is only provided as a service, not as downloadable software. Although free for the time being, Google reserves the right to add advertisements in later versions, and registration of a so called API key is necessary for every website using it. Google provides an API for integrating Google maps in custom web applications.

Apart from technical limitations the biggest problem with Google maps is that Google must be trusted to handle potentially sensitive data. This is simply not acceptable in a forensic application. There are also other online map services, but they have the same fundamental shortcomings as Google maps, and will therefore not be considered here.
**Google Earth**

Google Earth is a stand-alone GIS application, and not really a toolkit. However, Google has opened it up sufficiently to allow for some customization using the Keyhole Markup Language (KML) [kml06a, kml06b].

As with Google Maps and its equivalents an external company must possibly be trusted to handle potentially sensitive data. Google Earth continuously communicates with Google’s servers, and being a closed source application it is not possible to know what information is transmitted back to Google. Additionally it is only free for personal use, anything else incurs an annual fee [goo06b].

**GeoTools**

Geotools [geo06] is OGC compliant and open source. It is written in Java and currently is under active development. The newest version is 2.2.1, released on October 12 this year. GeoTools is partially based on JTS, described above. GeoTools fulfils some of our requirements, but it is lacking particularly with regard to useful documentation. The user guide is currently not in any consistent state and some of the Javadoc documentation is only available in French. We have had no success in getting test code running on geotools, at the time of testing there seemed to be a versioning conflict between libraries. This makes the learning-curve rather steep.

**OpenMap**

OpenMap is developed by BBN Technologies [ope06c]. It is based on Java and published as open source. The current version is 4.6.3, released this February 1. First made available to the public in 1998, OpenMap is a very mature toolkit, and is still actively maintained. OpenMap fulfils all of the requirements of Section 4.5.1, except that it is not OGC-compliant. Apart from this it is definitively the best all-round toolkit we have found. The API and documentation provided by OpenMap is clear and concise, and makes the toolkit easy to get started with.

### 4.5.3 OpenMap GIS Functionality

OpenMap uses projections of the Earth to perform its calculations and display the map on screen. A particular projection contains functionality for converting between screen coordinates and the latitudes and longitudes of the Earth for the given projection. The package `com.bbn.openmap.omgraphics` contains classes representing graphical objects that can be drawn on the map. The projections

---

4Version 2.2.0 was used for this as 2.2.1 had not been released yet.
correctly determine the representation of OMGraphics on the map. All OMGraphics contain a representation of their shape in the current projection in the form of \texttt{java.awt.Shape}. Using methods defined by \texttt{java.awt.Area} which implements \texttt{Shape} it is possible to perform intersections and other operations between the Shapes of OMGraphics while maintaining correct mappings between screen coordinates and Earth coordinates. This functionality is used in Section 4.2.1 to perform the necessary intersections between geographical constraints to acquire the intersection points of the confidence region used for further calculations.

Unfortunately OpenMap does not support OMGraphics wrapping the polar regions. Thus geographical constrains represented as OMGraphics from the two measurement nodes located on Svalbard in Chapter 5 having radii large enough to extend beyond the North pole is not supported in our application. OpenMap supports exporting the current map as an image. This is used to acquire the trace images used in the servlet GUI in Section 4.4.1.

4.6 Address Information Storage

As with user interaction and management the original MAFIF prototype left anything else than flat file storage to be implemented later. The reason for this was that any storage needs will necessarily be application specific. For the geolocation functionality we need to store information about the landmarks and about traced hosts. This information is not only stored for archival purposes, but actively used when new tracing operations are performed. The natural choice for a robust, durable and easily searchable storage solution is a database management system (DBMS). We have chosen to use HSQLDB, see Section 4.6.3.

4.6.1 Data Model

The database consists of three independent tables; Landmarks, Traced and Misc. The Landmarks table stores information about landmarks relative to the landmark at which the instance of the database is located, such as geographical distance and delay, as can be seen in Figure 4.12. The Traced table contains information about hosts already traced from the landmark in question. The Misc table contains a single record with information about the landmark’s location, how long since the delays to other landmarks were calculated and the latest bestline value. The data in the Misc table is stored in the database only for convenience. The data models for the Traced and Misc tables are available in Appendix B.3.2.

The program DBCreator and its helper class LandmarkReader is used to create and populate the database and tables at each host in the system, using the unicdb script.
TABLE LANDMARKS (  
    NAME VARCHAR(32) NOT NULL,  
    IPADR VARCHAR(39) NOT NULL,  
    CHECKED TIMESTAMP,  
    DISTANCE_KM DOUBLE NOT NULL,  
    LATITUDE DOUBLE NOT NULL,  
    LONGITUDE DOUBLE NOT NULL,  
    MIN_RTT DOUBLE,  
    AVG_RTT DOUBLE,  
    C1 DOUBLE,  
    EPSILON DOUBLE,  
    HASH VARCHAR(64),  
    PRIMARY KEY(NAME,IPADR)  
)

**Figure 4.12:** The logical data model of the Landmarks table.

### 4.6.2 Agent Connections to Database

Since our framework is in its entirety Java-based Java DataBase Connectivity (JDBC) is the natural choice for how the agents connect to the database [66]. Contrary to probing, database-connectivity is not separated into particular Java classes. Behaviours needing database access contain a method `connectDB()` that returns a connection to the database, this connection is then used for executing any queries needed by the behaviour in question.

The database is set up to allow only local connections, but access is nonetheless password protected. Currently the connection information is hard-coded into the `connectDB()` methods.

### 4.6.3 Database Software

JDBC lets any Java program connect to underlying databases in a product neutral way, as long as a JDBC driver for the database is available. Most databases provide a JDBC driver, a list of available drivers and supported databases is given in [jdb06]. So as long as the database supports JDBC we can focus on other requirements. As described above there will be an instance of the database running at every host participating in the system. Thus we need a lightweight and un-obtrusive database, that can run on almost any host without modifications to the hosts setup or the database software.

HSQLDB (HSQL) is a 100% Java based database [hsq06]. It is already used by JADE for (optional) persistent storage of the Directory Facilitator catalogue [67]. The qualities of HSQL make it very well suited to our purpose. Its low memory requirement, high performance and extensive SQL support coupled with it being a mature and well-tested product already used in conjunction with JADE, makes it
an ideal choice. HSQL is not multi-threaded, but it is multi-threading-safe. This is important as several agents may access the same information in the database simultaneously.

4.7 Limitations

To provide for scalability, both with regard to the number of simultaneous trace operations and the number of landmarks, several optimizations were planned, as well as implementation of the improvements and heuristics described in Section 3.5. Regrettably this has not been possible to accomplish in the available timeframe. The only optimization that has been implemented is the use of database lookups to avoid unnecessary measurements. Most of the unimplemented improvements would have been impossible to evaluate properly in the Uninett network, the test environment used in Chapter 5, due to few landmarks and the limited geographical extent of the network.
Chapter 5

Experiments

This chapter describes our test environment, experiments and results. All our experiments were carried out in the Uninett research network.

5.1 Environment - The Uninett Network

The Uninett research network links Norwegian education and science institutions and connects them to international research networks. As part of their infrastructure Uninett has deployed 15 measurement nodes to collect information on network performance. Our experiments were carried out using these measurement nodes, both by using already collected information and by gathering our own through the software described in Chapter 4.

5.1.1 Network Topology

The logical topology of the Uninett network, including link capacity, can be seen in Figure 5.1. Uninett rents most of the links from commercial providers. All links are optical, except for the connection between svalbard-mp and nyalesund-mp which is a 155Mbps radio link. Although the different physical links are rented from different providers, at the IP layer the network is configured as a single Autonomous System\(^1\). This is an important property, as there is virtually no path inflation in a single AS [54]. Route changes are fully distributed in about 3 seconds, and there is little to no route-flapping\(^2\).

\(^1\)Uninett uses a combination of IS-IS and iBGP-meshing for internal routing. Some customers use internal AS numbers and BGP-peering, but this does not affect our use of the network.

\(^2\)Route-flapping describes the behaviour of a router that advertises and withdraws reachability information, in quick sequence. It is caused by errors within the network, which might be in router configuration(s), links, software or hardware.
Figure 5.1: Logical topology of the Uninett network.

The geographic placement of the measurement nodes can be seen in Figure 5.2. Figure 5.3 gives the great circle distance, and the theoretical lowest delay limit between the measurement nodes. This limit is the same as the ideal baseline described in CBG in 4.2.1. The actual cable distances between the measurement nodes are of course not great circle distances, as discussed in 3.6.4. Also there is some overhead in routers along the path. Thus the real lower limit is higher than that given in Figure 5.3.

---

3The location markings in Figure 5.2 may have minor deviations from actual geographical location due to conversion from UTM coordinates to longitude/latitude, and inaccuracies in the image used for the landmass on the map. For more on this, see Appendix D.
5.1.2 One-Way Delay Measurements

As mentioned in Section 3.6.2 the usual way of measuring delay is to measure the RTT between two hosts, and not the actual one-way delay. The measurement nodes are equipped with GSM-synchronized clocks and passive measurement cards as part of the SCAMPI and LOBSTER projects [8], [lob06]. This makes it possible to measure one-way delay.

Correlation Between Distance and RTT

Figure 5.3 shows the correlation between halved RTT and lower limit one-way delay for the measurement nodes. On average the halved RTT is larger than lower limit one-way delay by a factor of 2.77. The largest difference is between grimstad-
mp and porsgrunn-mp where halved RTT is larger by a factor of 11.35. The difference for narvik-mp - svalbard-mp is the smallest with a factor of 1.48. This seems reasonable as the cable between Narvik and Svalbard runs mostly along the ocean floor and thus probably is relatively close to the great circle path between the two locations. Note that nyalesund-mp - svalbard-mp actually has a factor of 1, the halved RTT is identical to the lower limit one-way delay. As mentioned in Section 5.1.1 the connection between svalbard-mp and nyalesund-mp is a radio link. The lower limit one-way delays in Figure 5.3 are based on the speed of light in optical fibers, and this is clearly not correct for a radio link.

The correlation between RTT and great-circle distance is interesting with regard to how this is used in the CBG algorithm.
Actual One-Way Delay

To establish the actual one-way delay between some of the nodes in Figure 5.3 we used data gathered by Uninett using the passive measurement cards of the nodes. The data sets were gathered by sending 180,000 probes a day between October 1 and October 31. This gives an inter probe distance of 0.5 seconds. In analyzing the data sets we ran into some interesting problems. We originally wanted to calculate the one-way delay for the pairs Trondheim - Molde, Trondheim - Svalbard and Trondheim - Tromsø. However, the measurements from the node in Trondheim to the others consistently reported negative delay values. The relative fluctuations in the measurements for all pairs Trondheim - node$i$, node$i$ - Trondheim followed each other closely, so there was reason to suspect that the time of trd-mp was out of synchronization with the other nodes. It turned out that in fact only the nodes bergen, bo, grimstad, molde, narvik and tromso are GPS synchronized, the remaining use standard Network Time Protocol (NTP).

Due to the relative simplicity of the Uninett network topology the path from node A to node B is generally the same as the path from B to A. Therefore the one-way delays from A to B and from B to A should be very similar, and let us detect synchronization issues as the one described above. However, as described in Section 3.6.2, this is not the case for the Internet in general, and synchronization issues thus may be difficult to detect. This serves to illustrate why using one-way delay is difficult in practice. Unfortunately this also makes it difficult to give a good answer to if halved RTTs is a good approximation to one-way delay, and if the use of RTT in CBG and GeoBud leads to any loss of accuracy.

![Diagram](image)

(a) Alleged one-way delays in both directions between tromso-mp and svalbard-mp, showing an unprecedented degree of symmetry.

(b) True one-way delays in both directions between tromso-mp and narvik-mp, showing expected asymmetry.

**Figure 5.4:** Comparison of one-way delay measurements with and without GPS synchronization.

Upon closer inspection the relative fluctuations between the two oppositely directed one-way delays for many node pairs seemed too consistent. It turned out that
they actually were symmetrical. Figure 5.4(a) shows this for the delays captured October 24 between tromso-mp and svalbard-mp. The explanation for this is that the clocks of the nodes are not accurate enough without GPS synchronization at both nodes to actually measure values as small as the one-way delays. Thus an approximation of splitting RTT in half is used [RFC1305]. The alleged one-way delay measurements performed by Uninett are for most node pairs thus in fact not measuring one-way delays. True one-way delay between tromso-mp and narvik-mp is shown in Figure 5.4(b).

Figure 5.5: Correlation between halved minimum RTT and average one-way delay. This is very unfortunate as it makes the dataset for correlating halved RTT and one-way delay very small. The correlation is shown for available node pairs in Figure 5.5. Note that we have used average one-way delay, as the datasets provided by Uninett contain too much noise to reliably select a reasonable minimum value. The average correlation is 96.3%. For this limited dataset at least it seems that halved RTT is a good estimate of the one-way delay.

5.2 Test Setup

All tests were run using a computer at NTNU as the main container of the multi-agent system. This computer did not participate in actual delay measurements. This computer ran Ubuntu Linux v 6.06, JADE 3.4 and Sun Java 1.5.0_09-b01. All the measurement nodes ran Debian GNU/Linux 3.1, JADE 3.4, Sun Java 1.5.0_07-b03 and HSQLDB 1.8.0. All computers were X86-based. Tomcat 5.5.17, running on the NTNU computer, was used as the servlet container.

5.3 Limitations

Uninett is a production network, and we have little to no control of what other traffic is present at any time. To compensate for this all our experiments were run

---

4The computer was used in exactly one round of delay measurements, as the third host in the IPv6 vs IPv4 experiment.
several times, but there is no guarantee that we have experienced more than a subset of possible network conditions, and their effect on the results. This is however also the case when trying to locate any host in a real forensic situation. The sections below describe limitations particular to our use of Uninett as the environment for our experiments.

5.3.1 Measurement Node Traffic Types

The measurement nodes are part of Uninett’s production infrastructure, and are not really intended for anything else than doing a restricted set of traffic measurements. Due to restrictions on the types of network traffic allowed to some of the hosts the following measurement nodes were not able to take part in the multi-agent system: alta-mp, rena-mp, notodden-mp, bo-mp and porsgrunn-mp. Thus no measurements were performed from these hosts. They were, however, used as measurement targets in all experiments.

5.3.2 IPv6

Most of the differences between IPv4 and IPv6 described in Section 3.7 that affect the determination of geographical location are dependent on a relatively densely populated IPv6 address space to have any effect. This is far from the case currently, and it is therefore difficult to actually test the significance of these effects. The one exception is the difference in delay measurements between IPv4 and IPv6.

Unfortunately only two of the measurement nodes, narvik-mp and trd-mp, support IPv6 currently. As such use of CBG or GeoPing is pointless. However, we can extrapolate any differences in delay measurement between IPv4 and IPv6 for these two nodes, and come up with a factor to apply to all IPv4 measurements, thus creating an artificial data set for comparing GeoPing and CBG between IPv4 and IPv6.

5.4 Experiments and Results

Below the different experiments, what we hoped to prove through them and the results are presented. Where appropriate reference results calculated on the basis of Uninett data collected over a long period of time are used for comparison.

5.4.1 Varying Probe Parameters

For the measurement based approaches the accuracy of the delay measurements are very decisive for the final results. The number of probes and the distance between
them affect the total time of a trace operation. The more probes the higher the possibility of congestion for other hosts, and of detection by the target. Can we vary the different probe parameters to affect the accuracy?

All combinations of the following values have been tested. Number of probes: 3, 5, 10 and 100. Distance between probes: 0.2, 0.5, 1 and 2 seconds. By using the techniques described in Section 3.6.3 we analyze the different combinations.

**Results**

We have performed this experiment for all possible node pairs. The raw measurement data is available on the CD accompanying this thesis. Figure 5.6 shows the results for measurements from bergen-mp to all other nodes, the results from the other nodes are very similar.

![Graph](https://via.placeholder.com/150)

(a) RTT comparison with varied number of probes.

![Graph](https://via.placeholder.com/150)

(b) Underestimated distance constraints.

*Figure 5.6:* RTT comparison with varied number of probes and inter-probe distance.
The results in Figure 5.6(a) indicate that there is nothing to gain by increasing the number of probes. A very slight decrease in RTT values can be seen for the runs with more probes, but this decrease is less than 0.2ms on average between runs with 3 and 100 probes. The biggest difference is less than 0.4ms. This gives a correlation of 98.7% on average, and 97.5% for the worst case. Inter-probe distance, as shown in Figure 5.6(a), neither has any impact. The slight increase in the measured RTT to grimstad-mp for inter-probe delay 0.5 seconds must be attributed to coincidence.

During one of our test runs the link between Trondheim and Tromsø was down due to a severed optical fiber. In the period that the link was down all traffic along this distance was routed through the much lower capacity links Trondheim-Mo/Nesna-Bodo-Harstad-Tromsø and Trondheim-Narvik-Tromsø, see Figure 5.1. Uninett reports that the combined load on these two links were 90% during this period. Figure 5.7(a) compares the delays captured from tromso-mp to some of the other measurement nodes under normal conditions and when the Tromsø- Trondheim link was down. Figure 5.7(b) shows the corresponding values for C1, C2 and C3.

![Comparison of minimum RTTs](image-url)

![Comparison of C values](image-url)

**Figure 5.7:** Comparison of minimum RTTs and C values from tromso-mp under normal conditions and when the Tromsø-Trondheim link was down. Grey bars are values from when the link was down.

It is clear from Figure 5.7(a) that the downed link caused a partitioning of the Uninett network with regard to delay values and congestion. Hosts to which probes normally are routed along the Tromsø-Trondheim link show markedly increased delays. The increased C2 and especially C3 values in Figure 5.7(b) indicate substantial amounts of congestion. This is also the case for the hosts not included in the figures. We have chosen to show the data based on the runs with 100 probes, since the large number of probes results in more accurate C-values.

In contrast to when the Uninett network was in a normal state the difference between runs with different number of probes and inter-probe delay were larger when the link was down, as seen in Figure 5.8(a). The correlation between runs with 3 probes
and 100 probes is now 94.6% on average and 90.2% in the worst case. Notice that the nodes not having their routes changed by the downed link show more stable values.

(a) Comparison of minimum RTT values between runs with different number of probes.

(b) Comparison of minimum RTT values between runs with different inter-probe delay.

Figure 5.8: Comparison of minimum RTTs between runs with different probe parameters from tromso-mp when the Tromso-Trondheim link was down.

The picture is more complex with regard to inter-probe delay. There is no clear consistency in which inter-probe delay gives the best results in Figure 5.8(b). Also here the nodes not having their routes changes show stable values.

The larger differences between runs with different number of probes under difficult network conditions are interesting. Although the correlation between 3 and 100
probes is still strong, the difference is now of a size that can affect the accuracy of at least CBG if not GeoPing.

It is difficult to generalize these results to the entire Internet, but there at least seems to be a diminishing return in increasing the number of probes beyond 10. The cost of 90 more probes from every landmark to the target for an increased accuracy equivalent to that of going from 3 to 10 probes is not a reasonable trade-off.

### 5.4.2 IPv4 vs IPv6

Theoretically there should be a somewhat higher delay when using IPv6, see Section 3.6.2. Is it possible to prove this difference in Uninett? We perform several delay measurements between the same set of hosts equipped with dual networking stacks, using respectively IPv4 and IPv6.

#### Results

Table 5.1 shows the differences between IPv4 and IPv6 for measurements between the three hosts trd-mp.uninett.no, narvik-mp.hin.no and a computer connected to the campus network at NTNU dubbed localhost\(^5\). \(\varepsilon\) is left out for the host pair trd-mp - localhost since the technique used to estimate it is not accurate enough to give correct data for such small RTTs.

The RTT is consistently higher for IPv6 than for IPv4, as predicted in Section 3.6.2. Notice the relatively big increase in difference between narvik-mp - trd-mp and narvik-mp - localhost. This is probably due to the fact that packets to/from localhost must pass through at least one more router than those to/from trd-mp. The big difference between IPv4 and IPv6 for trd-mp - localhost support this assumption.

\(\varepsilon\) seems to be slightly higher for IPv6 than for IPv4, but the difference is too small to draw any definite conclusion with this small a data set.

According to Uninett some of their routers process IPv6 in software, this is probably much of the reason for the higher RTTs when using IPv6. We tried running measurements for this test when the Tromsø - Trondheim link was down, and got consistently much higher packet loss and delays for IPv6 than for IPv4. The big difference is probably caused by the fact that under high load the routers’ CPUs are already highly utilized and software processing of IPv6 suffers.

To extrapolate the IPv4 vs IPv6 results for use in GeoPing and CBG we used the average of the difference between trd-mp and narvik-mp in both directions. This

---

\(^5\)IPv4 address: 129.241.209.196
resulted in a factor 1.06. We applied this factor to existing IPv4 runs of GeoPing and CBG. For GeoPing the difference was too small to have any effect. A sample result for CBG is shown in Figure 5.9.

### Table 5.1: Comparison of RTTs between the three hosts using IPv4 and IPv6.

<table>
<thead>
<tr>
<th></th>
<th>trondeim-mp</th>
<th>narvik-mp</th>
<th>localhost</th>
</tr>
</thead>
<tbody>
<tr>
<td>trd-mp</td>
<td>minRTT</td>
<td>18.33 / 19.48</td>
<td>0.29 / 0.76</td>
</tr>
<tr>
<td></td>
<td>ε</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>narvik-mp</td>
<td>minRTT</td>
<td>18.33 / 19.55</td>
<td>18.47 / 20.05</td>
</tr>
<tr>
<td></td>
<td>ε</td>
<td>0.78 / 0.82</td>
<td>X</td>
</tr>
<tr>
<td>localhost</td>
<td>minRTT</td>
<td>0.28 / 0.76</td>
<td>18.46 / 19.85</td>
</tr>
<tr>
<td></td>
<td>ε</td>
<td>-</td>
<td>0.67 / 0.92</td>
</tr>
</tbody>
</table>

The increase in the estimated confidence region between the two confidence regions in is very small, and of little consequence for practical purposes. However, if the observation above that the RTT value increases markedly for every router a probe has to pass through, the difference compared to IPv4 should be larger for runs where the probes encounter many routers. Using GeoBud to estimate the buffer delays of the routers could probably mitigate this though.

### 5.4.3 Effect of Number and Placement of Landmarks

The number and especially placement of Landmarks is important to the accuracy of both CBG and GeoPing. GeoPing needs evenly spaced landmarks as it uses landmark locations as estimates of the location of the target. The constraint-based technique used in CBG suffers greatly when the target is not surrounded by landmarks. This was reported in [6] where the distance from the estimated target location and the actual location of the target varied from 21.5 km for a host.

---

As the servlet developed in Chapter 4 does not support multiple traces on the same map image Figure 5.9 was created using the stand-alone OpenMap application.
in central Europe to 562.4 km for a host in Umeå, Sweden. The confidence regions varied correspondingly, from 13,993 km$^2$ to 1,120,300 km$^2$. The host in Sweden was located to the north of most landmarks, and clearly demonstrated the reliance of CBG on landmark placement to provide accurate results.

The measurement nodes constitute a relatively limited set of landmarks. Nonetheless it is interesting to see if we can affect the accuracy of in particular CBG by varying which landmarks are used.

**Results**

Table 5.2 gives results for GeoPing using all measurement nodes. All the estimated locations are reasonable, with regard to the location of the target and the landmark selected to be most similar. It is interesting that rena-mp is calculated to be the landmark most resembling both oslo-mp and trd-mp. The reason for rena-mp being most similar to trd-mp is probably that the three measurement nodes to the north of trd-mp has lower RTTs to rena-mp than to molde-mp. In Table 5.3 where the three northern measurement nodes are not used molde-mp is the most similar to trd-mp, as expected.

<table>
<thead>
<tr>
<th>Target</th>
<th>Estimated Location</th>
<th>$\Delta(DV)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>narvik-mp</td>
<td>tromso-mp</td>
<td>11.9227</td>
</tr>
<tr>
<td>trd-mp</td>
<td>rena-mp</td>
<td>17.2234</td>
</tr>
<tr>
<td>bo-mp</td>
<td>notodden-mp</td>
<td>1.3367</td>
</tr>
<tr>
<td>oslo-mp</td>
<td>rena-mp</td>
<td>6.8543</td>
</tr>
</tbody>
</table>

*Table 5.2: GeoPing results using all measurement nodes*

<table>
<thead>
<tr>
<th>Target</th>
<th>Estimated Location</th>
<th>$\Delta(DV)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>trd-mp</td>
<td>molde-mp</td>
<td>7.5934</td>
</tr>
<tr>
<td>bo-mp</td>
<td>notodden-mp</td>
<td>0.9074</td>
</tr>
<tr>
<td>oslo-mp</td>
<td>rena-mp</td>
<td>5.1863</td>
</tr>
</tbody>
</table>

*Table 5.3: GeoPing results using measurement nodes south of and including trd-mp.*

The results for CBG are given in Table 5.4. These results are not very accurate. Due to the landmark locations and the number of landmarks this is to be expected. The most interesting result is that the location of the landmarks seem to be much more important than the total number of landmarks. Figure 5.10 shows this clearly. If not for the $C_{ir}$ of narvik-mp from the north, the confidence region would have been much larger. The multiple landmarks mostly to the south do not add much accuracy beyond the first. Not only the placement of landmarks relative to the target, but also the distance between target and landmarks, play an important part.
Figure 5.11 show this for a trace of bo-mp, where the inclusion of the $C_{tt}$ of oslo-
mp greatly decreases the confidence region.

<table>
<thead>
<tr>
<th>Target</th>
<th>Lat.</th>
<th>Lon.</th>
<th>Est. Lat</th>
<th>Est. Lon</th>
<th>$\Delta [km]$</th>
<th>$R [km^2]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>trd-mp</td>
<td>63.4141</td>
<td>10.4059</td>
<td>63.6919</td>
<td>10.1508</td>
<td>33.41</td>
<td>41 214</td>
</tr>
<tr>
<td>notodden-mp</td>
<td>59.5712</td>
<td>9.2606</td>
<td>60.0288</td>
<td>10.3635</td>
<td>80.06</td>
<td>97 200</td>
</tr>
<tr>
<td>bo-mp</td>
<td>59.4238</td>
<td>9.0661</td>
<td>60.0482</td>
<td>10.2121</td>
<td>94.68</td>
<td>93 312</td>
</tr>
<tr>
<td>narvik-mp</td>
<td>68.4360</td>
<td>17.4416</td>
<td>69.3231</td>
<td>18.1763</td>
<td>103.04</td>
<td>38 404</td>
</tr>
<tr>
<td>oslo-mp</td>
<td>59.9437</td>
<td>10.7174</td>
<td>60.6223</td>
<td>9.0449</td>
<td>119.26</td>
<td>20 615</td>
</tr>
</tbody>
</table>

Table 5.4: CBG results.

Figure 5.10: The best of the geolocation results using CBG.

Figure 5.11: Confidence region and estimated location of bo-mp.
5.4.4 CBG Overestimation Factor

The CBG algorithm uses RTT as an estimate of the double of the one-way delay. This fudge factor is used to ensure that underestimation will not occur, see Section 3.4.3. However, the value 2 and its approximation by using RTT, although very practical, substantially worsens the accuracy of CBG. Is it possible to determine a lower value to obtain more accurate results, while still avoiding underestimation? We try to determine a lower bound on this value, for use in the Uninett network. The result is not valid in any other network, but it might point to a lower value than 2 that may be generally used to improve the accuracy of CBG, without costly measures such as those proposed in GeoBud.

Results

This experiment did not yield the expected results at all. Underestimations occurred frequently when using the standard factor of 2. Figure 5.12 gives two examples. Note that inclusion of the $C_{i\tau}$ of oslo-mp in Figure 5.12(b) would not lead to an empty $R$, but produce an incorrect estimation. The corresponding situation arises in Figure 5.12(a).

The results indicate that, if anything, the factor should be increased for the Uninett network. Figure 5.3 shows that the correlation between lower-limit one-way delay and halved RTT is higher for host pairs exhibiting underestimation, than it is for host pairs that do not. We do not have data from other networks to compare the Uninett data to, but this might indicate that the Uninett network topology is more optimal than that of other parts of the Internet, where CBG seldom underestimates when using the default factor 2. The landmarks resulting in underestimation were not used in the other experiments. As described in Section 4.2.1 an algorithm for discarding the $C_{i\tau}$s of landmarks that leads to an empty $R$ should be implemented, as it would make CBG much more robust.

5.4.5 Moving Target

As discussed in Section 3.7 more and more Internet hosts are mobile. Is it possible, using existing techniques, to establish that a host has moved, and if so the direction of its movement? How large a distance must any movement represent to be detectable? For this experiment a laptop computer will be the target of several trace operations while being at different locations in Trondheim.

Unfortunately we do not have the time or resources to perform the moving target experiment at an adequate set of locations. This will severely limit the possibilities of establishing how large the distance between locations must be to be detected as a probable move. Three locations were used Nardo, Munkvoll and Dragvoll.
The network connections at all three locations were Asymmetric Digital Subscriber Lines (ADSL). The Munkvoll and Dragvoll connections were delivered by Telenor, while the Nardo connection was delivered by NextGenTel.

**Results**

The Munkvoll location proved to be unusable. CBG consistently returned constraints of well above 40 000 km, more than the circumference of the Earth. GeoPing did
not fare any better, with $\Delta(DV)$ values varying between 0.3 and 264 000. Clearly the network conditions made the use of these two techniques impossible.

The CBG-results for the Nardo and Dragvoll locations are shown in Figure 5.13. The results were relatively consistent over multiple runs, but without knowing the locations of the target beforehand it is impossible to determine if the location of the target actually changed. With confidence regions several times larger than the total area of Norway the results cannot be called accurate. The location estimates are off with about 292.3 km for Dragvoll and 61.8 km for Nardo. These values are clearly too large to determine if a host moved a distance of a few km.

The GeoPing results are shown in Table 5.5. These results also were consistent, without being useful for determining if the target’s location changed

Figure 5.13: CBG runs wit target located at Nardo and Dragvoll.
It is clear from these results that the larger delays incurred by non-optical-fiber networks is relatively poorly handled by GeoPing and CBG.

Table 5.5: Moving target GeoPing results

<table>
<thead>
<tr>
<th>Estimated Location</th>
<th>min $\Delta(DV)$</th>
<th>max $\Delta(DV)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dragvoll</td>
<td></td>
<td></td>
</tr>
<tr>
<td>molde-mp</td>
<td>31.1318</td>
<td>32.1125</td>
</tr>
<tr>
<td>porsgrunn-mp</td>
<td>32.3154</td>
<td>32.6606</td>
</tr>
<tr>
<td>Nardo</td>
<td></td>
<td></td>
</tr>
<tr>
<td>molde-mp</td>
<td>24.6418</td>
<td>30.0110</td>
</tr>
<tr>
<td>porsgrunn-mp</td>
<td>30.4451</td>
<td>34.6249</td>
</tr>
</tbody>
</table>
5.4.6 Scalability

The scalability of the geolocation application in MAFIF is interesting if it is to be used as a tool for law enforcement. Especially if a more refined version is to be employed on a much larger number of host, the ability to run multiple trace operations simultaneously will be important. The two most interesting metrics to explore is the increase in time elapsed and if the accuracy of the results are influenced.

We compare the time needed to complete 1, 5, 10 and 25 simultaneous trace operations, and if the accuracy in the results change with any significance. The results in Section 5.4.1 indicate that if we see any change in accuracy it is probably not due to externally caused change in network conditions, but rather self-interference. [40, 39] discuss the possibility of self-interference in delay measurements. That is, a host can send probes with an inter-probe delay so low that it causes congestions on the path it attempts to measure, and thus heavily influence its own measurements. Although we have chosen the default inter-probe delay to be well above the threshold of self-interference, with simultaneous trace operations probes are sent more often, and self interference may occur.

GeoPing was used for this experiment, as our implementation of it puts a higher load on the agent-system than the CBG implementation. When running simultaneous operations the CBG-bestline will be computed by the first instance only, all successive instances use a cached version. The most calculation intensive part of CBG is done outside the agent system, in the servlet, as explained in Sections 4.2.1 and 4.4.1. We are not interested in measuring the servlet scalability, but that of the geolocation functionality in MAFIF. Additionally the servlet uses a single Swing-thread for serving all requests, and is necessarily limited by this. GeoPing is also more sensitive to varying delay measurements, as it may result in the selection of entirely different landmarks as estimated locations. In CBG any small variations will only impact the size of the confidence region.

Results

Table 5.6 shows the landmarks consistently returned by all instances when narvik-mp was used as the target. Note that svalbard-mp did not perform measurements. This is the reason that nyalesund-mp is not in the list of estimated locations. To all other landmarks it appears to not be close to narvik-mp, while in the measurements of nyaleund-mp svalbard-mp appears to be close. With more landmarks the high values of a single landmark in Equation 4.11 would not have such a big impact. The $\Delta(DV)$ values listed are the minimum and maximum of three runs. It is quite possible that self-interference will make an impact with a higher number of simultaneous instances.
<table>
<thead>
<tr>
<th>Estimated location</th>
<th>min $\Delta(DV)$</th>
<th>max $\Delta(DV)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nr of simultaneous instances: 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tromso-mp</td>
<td>10.75</td>
<td>-</td>
</tr>
<tr>
<td>svalbard-mp</td>
<td>39.56</td>
<td>-</td>
</tr>
<tr>
<td>trd-mp</td>
<td>44.86</td>
<td>-</td>
</tr>
<tr>
<td>Nr of simultaneous instances: 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tromso-mp</td>
<td>10.81</td>
<td>11.21</td>
</tr>
<tr>
<td>svalbard-mp</td>
<td>38.31</td>
<td>39.52</td>
</tr>
<tr>
<td>trd-mp</td>
<td>44.66</td>
<td>44.69</td>
</tr>
<tr>
<td>Nr of simultaneous instances: 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tromso-mp</td>
<td>10.76</td>
<td>11.83</td>
</tr>
<tr>
<td>svalbard-mp</td>
<td>39.14</td>
<td>39.65</td>
</tr>
<tr>
<td>trd-mp</td>
<td>44.56</td>
<td>45.72</td>
</tr>
<tr>
<td>Nr of simultaneous instances: 25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tromso-mp</td>
<td>10.76</td>
<td>11.83</td>
</tr>
<tr>
<td>svalbard-mp</td>
<td>38.48</td>
<td>39.78</td>
</tr>
<tr>
<td>trd-mp</td>
<td>44.56</td>
<td>45.72</td>
</tr>
</tbody>
</table>

Table 5.6: Variances in $\Delta(DV)$ between different number of simultaneous instances.

Timing was done manually, and as such is not accurate more than to the second. The average times to complete all instances were about 30 seconds, independent of the number of instances. This is a marked improvement over the scalability results in Appendix A. This is probably due to the workload of the geolocation functionality being less than that of the content securing application used in Appendix A. Much of the elapsed time is spent idle waiting for probe packets to return. The content securing application is dependent on disk I/O performance and also spawns up to hundreds of agents. Also the optimization described in Section 4.1.1, resulting in fewer messages exchanged between agents, probably influences the result positively.
Chapter 6

Conclusions

We have successfully implemented geolocation functionality in MAFIF, showing that MAFIF indeed can be used as a general framework for Internet forensics. The limited scalability testing is promising, and shows a marked improvement over the previous MAFIF version.

We have also analyzed current geolocation techniques, with respect to the important properties accuracy, effort, reliability and the possibility of detection by the target. Several possible improvements to these techniques have been described, although we have not had the possibility of testing the impact and practical feasibility of all the improvements. The results of the experiments in Chapter 5 show that improvements are needed. The current geolocation algorithms have shortcomings that severely influence accuracy, especially when landmark placement is not optimal. The proposal to use dynamic regions set forth in Section 3.5.2 is probably the single improvement best suited to address this.

The experiments with varying probe parameters indicate that the Uninett network is very stable, and as such may not be a representative environment to gather knowledge of delay properties and variations over time, with regard to the Internet in general. However, a compromise must be made between having a relatively controlled, and at the same time sufficiently complex test environment.

Further contributions include the analysis of the impact of IPv6, and the introduction of multi-party computation to geolocation. The extensive focus on delay measurements, although not bringing anything new to the field of networking in general, is also new to geolocation as far as we know.
Chapter 7

Further Work

We have demonstrated that the framework for Internet Investigations we designed and built in [1] is indeed extensible and scalable. However, it remains a prototype and there is still much to be done with regard to utilizing multi-agent technology in the context of Internet forensics, particularly with regard to geographical location of Internet hosts. The sections below describe possible areas of further work.

7.1 Large-scale Experiments

Internet-wide experiments with a refined version of the geolocation application is an obvious next step. This should include implementing support for other protocols for probing than ICMP, and the optimizations and improvements described in Sections 3.5.1 and 3.5.2. Particularly determining the feasibility and effect of the dynamic region scheme would be interesting. Implementing GeoBud and other similarity models for GeoPing would allow for direct comparison with CBG and the current Euclidean distance-GeoPing. Determining if a smaller overestimation factor for CBG for general use is feasible, possibly at the cost of discarding some measurements, would also be of interest.

7.2 Multi-Party Computation

In Section 3.5.3 we described how geolocation can be augmented by multi-party computation. Adapting the GeoPing and CBG algorithms to use multi-party computation would open up new possibilities for cooperation by limiting the degree of trust necessary, while doing away with the limitations in Reistad’s proof of concept. The use of multi-party computation increases the necessary information exchange
between nodes. Assessing the scalability implications of this is important to determine the viability of using multi-party computation in geolocation.

### 7.3 IPv6

The limited experiments of Chapter 5 on IPv4 vs IPv6 indicated that the differences in the Uninett network were not particularly large. There may be more significant differences in other parts of the Internet. Conducting more extensive experiments to make clear the state of these differences would be useful. Also, mapping the extent of HMIP usage, and its actual effect on geolocation would contribute important information on how to deal with increased host mobility.

The possibility of frequently changing and random addresses described in Section 3.7.1 may make current geolocation techniques practically obsolete if widely adopted. Research into novel approaches to geolocation that can counter this would be extremely useful, even if this scenario should not come to pass.

### 7.4 Detection of Direction of Movement

To be able to detect the mobility of a host is very interesting. The experiment of Section 5.4.5 was not successful in doing so at all. This clearly indicates that more accurate techniques than currently available are needed. Determining the current location of a mobile host is not necessarily very useful, as the information is potentially quickly outdated. If accuracy could be increased enough to allow successive trace operations to establish a pattern of movement on the other hand, future locations could be estimated with some probability.

### 7.5 Web Service Integration

Making the system accessible as a web service through the use of one of the web service integration possibilities mentioned in Section 4.4 would allow for a more flexible integration of MAFIF in existing systems. It could also make it possible for MAFIF applications to make use of functionality from other web services.
References


[32] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for


RFC References


Floating References


[Dea06] Denis J. Dean. The universal transverse mercator (UTM) coordinate system. http://www.warnercnr.colostate.edu/class_info/nr502/lg3/datums_coordinates/utm.html part of the subject NR502 Geodesy, Cartography and Map Reading at the Geospatial Sciences program in the College of Natural Resources at Colorado State University, September 2006.


Appendix A

Article

This appendix contains a draft for an article based on the work we did in [1]. It is included here to give a concise introduction to the multi-agent framework for Internet investigations that this project is based on. The article will be expanded to include the geolocation functionality developed in this thesis and submitted to the Usenix Security'07 conference.
A Multi-Agent Framework for Internet Investigations

Abstract
With a dramatic increase of Internet related crimes, the field of Internet investigations is becoming increasingly important. Available tools and methods for Internet investigations are limited in scope, and they often do not sufficiently protect and document the integrity of digital evidence. In this paper, we introduce a novel approach to Internet investigations using multi-agent systems, providing the necessary scalability and security for large scale Internet investigations. Central aspects of the proposed approach are the preservation of evidence integrity and the mitigation of unwanted detection by investigation targets. A framework for multi-agent Internet investigations with a proof-of-concept application for securing digital evidence from web sites is implemented. Experiments using the proof-of-concept application show promising results with regards to scalability and generated traffic patterns.

Keywords: Internet Investigations, Multi-Agent Systems, Digital Forensic Science.

1 Introduction

With the immense success of the Internet as a global marketplace interconnected with virtually all aspects of physical life, cyber crime and fraud on the Internet is becoming a major issue, and even traditional criminal cases often involve digital evidence on the Internet. The media reports that Internet-related crime now out-paces drug trafficking in the United States, as measured in turnover [19]. Law enforcement has not been able to fully keep up with the explosive growth, making the Internet an attractive arena for criminals seeking the anonymity of the masses. In order to effectively handle this development, a new generation of tools for Internet investigations is needed. If a crime scene has a digital component on the Internet, we need forensically sound tools for Internet investigation that follow the same rigorous standards as those set for traditional forensics.

The current generation of tools is limited and immature [16, 17]. Most of these tools are based on clients running on single hosts, and many popular tools do not sufficiently protect and document the integrity of the digital evidence. The centralized tools have limited scalability and load-distribution, and the reliance on a static address makes the systems vulnerable to detection by criminals targeted for investigation. Based on the results and recommendations published in [15, 9, 10], we aim to partly mitigate these problems and to help establish a framework for a new generation of investigation tools powered by multi-agent technology. We address this by proposing a prototype framework for Internet Investigations using the JADE agent development framework [3]. The Internet Investigations framework is designed to support rapid development of a number of applications, such as securing content from different services and protocols on the Internet, tracing of IP addresses, and monitoring networks for user information and traffic patterns. Such an application concept is shown in Figure 1.

Section 2 discusses related work. Benefits and challenges of using multi-agent systems in Internet Investigations are identified in Section 3. Section 4 describes the prototype framework, while
Section 5 presents experimental results for the proof-of-concept application. Section 6 concludes this paper and outlines areas for further work.

2 Related Work

The use of distributed systems for gathering and processing security-related information has been extensively explored in the field of Intrusion Detection Systems (IDS). The idea of using agents in IDSs was introduced by Crosbie and Spafford in [7, 8] and further explored in [2]. The latter describes existing IDS architectures and their limitations and provides an analysis of how a system of autonomous agents can help in overcoming these challenges. Carver et al. take the use of agents a step further, and propose an integrated methodology for both intrusion detection and response [14]. An IDS based on a combination of stationary and mobile agents travelling the network is presented by Helmer et al. in [12]. Wei describes a more comprehensive system, where mobile agents integrate with existing firewalls, honeynets and other IDS [23]. A simple framework for distributed network forensics is presented in [20]. This framework employs IDS as a means of detecting attacks, but the main purpose is not to prevent or warn about attacks, but to collect enough evidence to perform an investigation. Our approach to using multi-agent systems for Internet investigations takes a similar architectural approach as these systems, but the application area and functionality of the agents overlap only to a small extent. Our system is designed to support Internet Investigations in a forensically sound fashion, involving a wide range of services and protocols. We also employ multi-agent systems not only for scalability and performance reasons, but also for reducing the probability of detection and for handling increasingly dynamic technologies, such as P2P.

3 Multi-Agent Systems Applied to Internet Investigations

The distributed architecture of a multi-agent system has many properties that are of immediate use to the discipline of Internet Investigations. Below we discuss the possibilities these properties open up for.

3.1 Traffic Patterns and Detectability

A shortcoming of the existing systems is their vulnerability to detection due to the generation of large amounts of traffic from a single host. Using a multi-agent system for securing the content of
a website or file server can result in a less obvious traffic pattern, due to several agents at different locations sharing the load, and not downloading everything in one session. If a website is to be secured repeatedly the set of agents and hosts participating in the action can be changed, so as not to generate similar traffic to the same set of addresses. The dynamic nature of the system could be enhanced further by allowing any host/address to be part of the system only for a limited amount of time, or by using a schedule with long periods of inactivity per host/address.

Networks of Internet sensors used for detecting malicious traffic have been shown to be vulnerable to mapping attacks, based on publicly available information [4]. Applications using blacklists for blocking known ‘suspicious’ addresses are already in use. Should the addresses of a system used for Internet Investigations be included on such a list, the usefulness of the system could drop dramatically. A dynamic approach like the one described above leaves the system less vulnerable to attempts at discovering its existence and mapping its extent. Although less likely, the traffic pattern generated using such an approach can still be detected using passive fingerprinting.

Having multiple agents at different locations may also make it possible to pose as users in P2P systems. Agents participating in file sharing networks could gather information about the files being shared, IP addresses and activity patterns of other users and other information depending on the network and protocol in question.

### 3.2 Scalability, Load-Balancing and Redundancy

Another shortcoming of the existing systems is the reliance on public information sources and services. These sources and services often apply access restrictions, limiting the number of connections from any host in a given period [9, 10]. In a multi-agent system, requests to such sources and services can be done in a round robin fashion, distributing the access requests, thus lowering the probability of the system being denied access. This will allow the system to run multiple sessions requesting such information simultaneously. Additionally a distributed multi-agent system need not be dependent on any single host, and may recover more gracefully from communication breakdowns or the failure of a number of hosts. Communication breakdowns and host failures can be mitigated by agents at other hosts taking over the workload of the disconnected or lost agents. Another approach is to periodically save the state of all critical agents, and if anything happens to the containing host(s), create new agents and load the saved state at operative hosts.

### 3.3 Geographical Location

In a multi-agent system it is possible to use the hosts participating in the system itself as landmarks, instead of using publicly available Looking Glass hosts as landmarks and dealing with the accompanying problems, as in [11]. These hosts can be used exclusively or in addition to publicly available hosts. In this way it is possible to pinpoint the location of the landmarks using for instance the Global Positioning System (GPS). One would also probably have better knowledge of the physical layout of the network connecting the hosts, and thereby be better equipped to calculate deviances between network distance and actual distance between landmarks. Using the same software across all hosts also makes it possible to standardise the format of requests and replies.

### 3.4 Adaptability

The benefits achieved by using a multi-agent system described in the preceding sections are largely due to the distributed nature of the system, and not because it is agent-based. A multi-agent system
provides well-defined interfaces, and mostly independent components in the form of agents. This is in itself valuable, but the real power of agents lie in their adaptability and autonomy. The Internet is not a static environment, P2P networks in particular have a high rate of change. Adaptability is a very useful property in this situation. It might be difficult to foresee all possible events in such a complex environment, combined with autonomy adaptability is a good solution to this problem.

3.5 Challenges of Distribution to Forensic Soundness

The use of a multi-agent system also introduces some challenges. The system in itself is more complex, and is subject to all the challenges of distributed systems, often referred to as the fallacies of distributed computing. This extra complexity of distributed systems makes it harder to maintain the chain of custody, as there is no longer a single host-operator pair acting as custodian at all times. Several hosts may be involved, and the possibility of one of them becoming compromised is very real. Precautions must be taken so that a single compromised host does not void the forensic soundness of the rest of the system. If one or more hosts become unavailable and contain agents that are in the middle of investigation sessions the system should be able to cope by redistributing the tasks of the affected agents to other agents, and maintain the integrity of the ongoing investigation.

Even when all hosts of the system are available, and with no malicious activity, some challenges remain. As part of the chain of custody it is important to correctly date the digital evidence. When agents at different hosts more or less simultaneously collect evidence from the same digital crime scene, they need to access synchronised and secure time. This is important to be able to prove the relation of different pieces of evidence, i.e., that they existed in the given crime scene at the same point in time [13].

It is inherent in Internet Investigations that active methods can change the state of targeted systems. It is consequently important that investigative tools keep detailed logs for the purpose of documenting all connections. Such documentation is essential for the purpose of presenting digital evidence in court.

4 Framework Prototype

We have developed a prototype framework for multi-agent based applications for Internet Investigations, capable of running multiple simultaneous investigation processes. The forensic soundness is upheld by maintaining a chain of custody, using extensive logging, checksumming and encryption of agent communication.

4.1 Underlying Multi-Agent Platform

There exist many platforms for developing multi-agent systems. We have chosen JADE [3], due to its FIPA\(^1\) compliance and extensive use in multi-agent research. In addition, although JADE is not designed with scalability as its main goal, it scales rather well [5, 6]. JADE uses messages based on the FIPA Agent Communication Language (ACL) and ontologies\(^2\) for agent communication. All ontology concepts are represented as Java classes. We have defined an ontology dealing with Internet Investigations, using the ontology editor Protégé [1]. The classes were then generated by a JADE plugin for Protégé named Ontology Bean Generator [21].

\(^1\)Foundation for Intelligent Physical Agents, an IEEE Computer Society standards committee.

\(^2\)Ontology: The conceptualisations that the terms of a knowledge representation vocabulary are intended to capture, about the world or more often a specific domain.
All communication between agents on different containers is encrypted, and all communication, whether intra- or inter-container, is signed. This is done through a security add-on to JADE named JADE-S. JADE-S also provides functionality for ownership control, and access control based on this. Taking advantage of this functionality would make it possible to give individual hosts and users different access privileges, and to a degree enforce accountability. This has not been implemented in the prototype, but would be critical in a large-scale production deployment.

### 4.2 Agent Design

Due to the strict requirements of collecting and preserving digital evidence, it is not desirable that the agents act too autonomously. All parts of the process must be controllable by a human operator, or at least it must be possible for an operator to verify the steps in detail afterwards. However, a limited degree of autonomy is desirable. Once an agent is given a task by an operator we would like the agent to decide by itself how to best accomplish the task. That is, how many other agents it needs to cooperate with, and how the task is divided into subtasks and shared between the participating agents. To allow for this limited autonomy we have created five different agent types, each with their specific roles. For each container there are single, non-transient AdminAgents, LogAgents and TimeAgents. Transient SessionAgents are created by AdminAgents when needed, the SessionAgents in turn can create WorkerAgents. The different agent types, identified by the first letter in their type name, are shown in Figure 2.

The AdminAgent agent type is the operator’s single point of entry into the system. It is provided with several behaviours to handle incoming requests, schedule (future) executions and report on or abort any of its investigations, see Figure 3(a). The AdminAgent does not execute any investigations itself, to do this it creates one or more SessionAgents in its home container that handle(s) the current session(s) of the investigation. This is necessary for the AdminAgent to be able to speedily handle operator requests, and manage multiple (active) investigations. The operator may request a specific AdminAgent, or a heuristic for choosing one may be developed, based on geographical location, host load or any other (combination of) metrics.

The SessionAgent agent type is, as the AdminAgent, equipped with behaviours for handling requests, aborting and reporting. No scheduling is performed by the SessionAgent, instead it may be equipped with different application specific initiation behaviours for preliminary execution of investigation sessions. Which of these initiation behaviours is executed depends on the type of investigation. Generally such a behaviour performs just enough of the investigation necessary to create a set of WorkerAgents and distribute work items to them. The distribution of WorkerAgents among the containers of the system is decided by the initiation behaviour, depending on the type investigation. An example distribution is shown in Figure 2, where greyed out agents belong to...
Instead of initiation behaviours the WorkerAgent agent type may be equipped with different behaviours for executing assigned work items. A WorkerAgent instance is limited to a single work behaviour, but only this behaviour needs to be exchanged or modified for the agent type to support different types of work items. This makes it possible to deploy a new Internet Investigations application by modifying only this behaviour in addition to a SessionAgent initiation behaviour. When a WorkerAgent is done with its assigned work item, or unable to complete it, it typically reports to its SessionAgent and terminates. A session is finished once all its WorkerAgents are terminated. The SessionAgent then reports to its AdminAgent and terminates. The AdminAgent closes the session by logging its completion status to the session log.

The TimeAgent agent type is responsible for keeping the time of its local container synchronized to a trusted time source. The prototype implementation uses a behaviour that queries a pool of NTP servers. The time source can be switched by altering this behaviour, without affecting the rest of the agent type. It periodically queries the configured time source and adjusts the local time accordingly. If the offset between the trusted time source and local time is above a given threshold it is logged to the system log, and all local agents are informed of the adjustment. Additionally all agents may query the TimeAgent at any time for an offset to their local time. All TimeAgents are registered with the Directory Facilitator such that if an agent is unable to contact its local TimeAgent it can easily query another.

**Figure 3:** The architecture of a JADE agent and the agent as part of the JADE distributed agent system, adapted from [3].

The LogAgent agent type is responsible for all logging. In the prototype system two types of logs are kept, a session log and system log. System logs contain information related to the system itself, like time changes and agent failures. Session logs contain information about sessions like time stamps, participating agents and file hash values. Logging is done via the standard Java logging facility. A special Handler is used to forward all LogRecords to either the local LogAgent of the originating agent, or if the agent belongs to a non-local investigation, to the LogAgent local to the SessionAgent managing the investigation. Currently the LogAgents write the logs to XML-formatted files, but any type of non-volatile storage may be used.

The system can contain an arbitrary number of containers, on an arbitrary number of hosts, see Figure 3(b). The agent organisation resembles the hierarchic organisation described in [18], with the exception that new agents routinely are added.
5 Experimental Results

A proof-of-concept application for securing static content from websites was developed on top of the framework to demonstrate some of its functionality, and to show possible traffic patterns generated by a distributed Internet Investigations application. The application was implemented as a special agent and two behaviours added to SessionAgents and WorkerAgents, as explained in Section 4.2. The test application agent represents a human operator giving input to the agent system. It does this by sending an ACL message to an AdminAgent it finds querying the Directory Facilitator of the platform, requesting the AdminAgent to start a new investigation session based on the received URL. The application saves the first two levels of textual content of the website represented by the URL. MD5\(^3\)-sums are generated for every downloaded file, and is logged together with the location of the file, as part of upholding the chain of custody.

Three live web sites on public networks were used for testing. A simple static file-based host running Apache 2.0.54 set up by us, a university web site and an online newspaper. Site references have been anonymized, and we will refer to the sites as Simple, University and Newspaper. Due to using live websites the content and the network conditions may change over time. The hosts participating in the agent system running the test application will be referred to as Host A, B, C and D, for the same reasons of anonymity. To keep the test application simple all tests were run with the JADE Remote Management Agent GUI enabled. This slows things down to some extent, due to extra communication between the agent system and the GUI. In all tests correct checksums and timestamps were generated and added to the session log.

5.1 Traffic Patterns

The purpose of this test is to show possible traffic patterns generated by a distributed Internet Investigations application. The test application, running on hosts A to D, was set to secure the content of the Simple, University and Newspaper sites, one at a time.

![Figure 4: Number of connections to the webserver per host when running the different test cases.](image)

As can be seen in Figure 4 no host exhibit a significantly different number of connections than the others. This may help avoid detection by server operators, as described in Section 3.1. Controlling the exact number of connections per host is difficult, due to not knowing the link structure and number of files to secure beforehand. Figure 4 reflects this in that no host stand out as the one with the most or fewest connections across the test cases. No attempt has been made at avoiding duplicate downloads or balancing the distribution of connections in the test application. This results in the number of connections being higher than the actual number of files to download,

\(^3\)Recent research has uncovered weaknesses in the full MD5 [22].
due to duplicate downloads. In a production setup functionality for evening out the number of connections between participating host should be included.

5.2 Load Balancing and Scalability

Load balancing among the hosts in the system is important to avoid any host acting as a bottleneck to the whole system. Scalability with regard to the number of active agents and the number of simultaneous investigation sessions is also important. This is a good indicator of the scalability of the framework, not only the content securing application. Table 1 shows details of single runs for each of Simple, University and Newspaper.

<table>
<thead>
<tr>
<th>Target</th>
<th>Simple</th>
<th>University</th>
<th>Newspaper</th>
</tr>
</thead>
<tbody>
<tr>
<td>End status OK/failed</td>
<td>All OK</td>
<td>161/234</td>
<td>199/577</td>
</tr>
<tr>
<td>Runtime</td>
<td>16 sec</td>
<td>112 sec</td>
<td>83 sec</td>
</tr>
<tr>
<td>WorkerAgents per host/ total</td>
<td>10+10+11 / 31</td>
<td>17+17+16+19 / 69</td>
<td>38+43+43+64 / 188</td>
</tr>
<tr>
<td>Avg. files secured per WorkerAgent</td>
<td>3.77</td>
<td>2.33</td>
<td>1.06</td>
</tr>
<tr>
<td>Max files secured by a WorkerAgent</td>
<td>18</td>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td>Min files secured by a WorkerAgent</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Avg. time to secure a file</td>
<td>0.14 sec</td>
<td>0.70 sec</td>
<td>0.42 sec</td>
</tr>
<tr>
<td>Checksum and timestamp status</td>
<td>All OK</td>
<td>All OK</td>
<td>All OK</td>
</tr>
</tbody>
</table>

Table 1: Results of single runs in the three test cases.

The tests in Table 1 were run at a different time than those in Figure 4 and 5, and thus direct comparison is impossible. Nonetheless it is interesting that although the load division between the WorkerAgents is not very good, this does not result in correspondingly poor balancing in the number of connections per host. The field "End Status" in Table 1 indicates whether all attempted links resulted in secured files. The reason for the relatively large numbers of failed links in the University and Newspaper tests is a limitation in the link extraction code of the test application. Being a proof-of-concept the application does not handle dynamic link structures such as those generated by technologies like JavaScript or Flash. The initiation behaviour responsible for distributing work items is over-cautious and creates a very high number of agents, which results in few files secured by each agent. The total runtime and average time to secure a file does not appear to suffer from this, and indicates that the framework handles a relatively large number of agents well.

To stress the system, and show that it is actually capable of running multiple simultaneous investigation sessions, the test application was modified to send the request to secure the web site to all of the AdminAgents of the platform. This was done for the two most complex test cases, University and Newspaper, in turn. The results are shown in Figure 5. The lines marked by squares in each subfigure represents the time needed to run the same number of test cases serially, and are included for comparison.

![Figure 5: Elapsed time for sets of sequential vs parallel test runs of content securing.](image-url)
The data in Figure 5 is limited, the maximum number of simultaneous investigation sessions is 4. Up to and including this number of sessions the test application scales linearly. In the case of 4 sessions in Figure 5(b) the number of active WorkerAgents is about 1100. As mentioned above this is an excessive number. Modifying the initiation behaviour to distribute more work to each WorkerAgent would drastically reduce this number, and result in less overhead associated with agent life-cycle management for the agent system. The results in Figure 5, being from an unoptimized application, bodes well for the scalability of the framework. More extensive testing is needed to ascertain how well it scales with a massively increased number of simultaneous investigation sessions and more hosts.

6 Conclusions and Further Work

We have shown that the value of using multi-agent technology in the field of Internet Investigations is real, and that it indeed has the potential to help establish a basis for a new generation of investigation tools. We stated in the purpose of this project that we aimed to address the limitations of current tools and methods, related to scalability, evidence integrity, and unwanted detection by investigation targets. As our tests have shown, the idea of distributing the traffic over several hosts works, resulting in less detectable traffic patterns. The results in Section 5.2 shows this using only four hosts. By employing a larger number of hosts in different address ranges, the traffic pattern would be yet more difficult to detect. The agents could also be programmed to mimic the behaviour of human users in order to camouflage investigations and avoid passive fingerprinting attempts.

With regard to scalability more testing is necessary, but the initial results are promising. There is room for optimising performance, as this is only a prototype. The overhead incurred by the use of JADE-S is not known, and recent scalability analysis of JADE does not take JADE-S into account [5, 6].

The framework prototype is based on a mature and well featured multi-agent platform, and as such should be relatively easy to extend with desired functionality. The use of automated tools for ontology creation enforces consistency, and although we have tried to make the ontology flexible enough to handle extended functionality as it is, makes altering it a well-defined exercise. Compared to traditional object oriented designs, the agents is more independent of each other, and the high level communication using ACL messages makes interaction easy to grasp.

To further test the suitability of the framework for its intended purpose, additional Internet investigations applications should be implemented on top of it. Of particular interest would be geographical location with the landmarks as part of the multi-agent system, and agents acting as users in different P2P-based file sharing networks. Naturally, performance evaluations including measuring against other systems like GNU wget should be conducted. Extended scalability testing in particular would be of interest. Establishing the framework’s security is important in relation to the legally required integrity and confidentiality of the collected digital evidence.

Article References


[19] Souhail Karam (Reuters). Cybercrime yields more cash than drugs. In several major online newspapers under slightly different headings, November 2005.


Appendix B

Design Diagrams

This appendix contains the (A)UML diagrams for the classes used to implement the geolocation functionality in MAFIF. Some classes belonging to MAFIF but not developed as part of the geolocation functionality are also shown where they contribute to the understanding of the other classes and their roles. These classes are clearly marked with an $M$. The diagrams for all MAFIF classes are available in [1].
B.1 (A)UML Diagrams

B.1.1 AdminAgent

![UML Diagram of AdminAgent](image)

Figure B.1: The AUML class diagram of AdminAgent and its behaviours.
B.1.2 SessionAgent

---

Figure B.2: The AUML class diagram of SessionAgent.
Figure B.3: The AUML class diagram of SessionAgent’s behaviours.

```java
public class SessionAgent {
    // AUML class diagram
    public class Behaviours {
        public class TraceReduceBehaviour {
            public void traceReduceBehaviour(Agent a, Message m) {
                // Method implementation
            }
        }
        public class InitiateTraceSessionBehaviour {
            public void initiateTraceSessionBehaviour(SessionAgent s) {
                // Method implementation
            }
        }
    }
}
```
B.1.3 WorkerAgent

Figure B.4: The AUML class diagram of WorkerAgent.
Figure B.5: The AUML class diagram of WorkerAgent’s behaviours.
B.1.4 GWAgent

Figure B.6: The AUML class diagram of GWAgent.
Figure B.7: The AUML class diagram of GWAgent’s behaviours.
B.2 Servlet UML Diagrams

B.2.1 Servlet Core Classes

![Servlet Core Classes UML Diagram]

Figure B.8: The UML diagram of the core servlet classes.
### B.2.2 Servlet Alpha Classes

#### OMAAlphaCircle

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>+OMALphaCircle( center : LatLonPoint, radius : float, units : Length, nverts : int, alpha : float )</td>
</tr>
<tr>
<td>+setAlpha( alpha : float ) : void</td>
</tr>
<tr>
<td>+fill( g : Graphics ) : void(JavaAnnotations = @Override)</td>
</tr>
</tbody>
</table>

#### Alpha

<setter> +setAlpha( alpha : float ) : void

#### OMAAlphaPoly

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>+OMALphaPoly( llPoints : float[], units : int, lType : int, alpha : float )</td>
</tr>
<tr>
<td>+setAlpha( alpha : float ) : void</td>
</tr>
<tr>
<td>+render( g : Graphics ) : void(JavaAnnotations = @Override)</td>
</tr>
<tr>
<td>+setGraphicsForFill( g : Graphics ) : void(JavaAnnotations = @Override)</td>
</tr>
<tr>
<td>+setGraphicsColor( g : Graphics, paint : Paint ) : void(JavaAnnotations = @Override)</td>
</tr>
</tbody>
</table>

**Figure B.9:** The UML diagram of the servlet alpha classes.
B.2.3 Servlet Circle Classes

These classes are only used for zooming functionality and contain much unused legacy code.

Figure B.10: The UML diagram of the servlet circle classes.
B.3 Database

B.3.1 UML Class Diagram

![UML class diagram of database related classes.](image)

*Figure B.11:* The UML class diagram of database related classes.
B.3.2 Database Tables

TABLE LANDMARKS (  
    NAME VARCHAR(32) NOT NULL,  
    IPADR VARCHAR(39) NOT NULL,  
    CHECKED TIMESTAMP,  
    DISTANCE_KM DOUBLE NOT NULL,  
    LATITUDE DOUBLE NOT NULL,  
    LONGITUDE DOUBLE NOT NULL,  
    MIN_RTT DOUBLE,  
    AVG_RTT DOUBLE,  
    C1 DOUBLE,  
    EPSILON DOUBLE,  
    HASH VARCHAR(64),  
    PRIMARY KEY(NAME,IPADR)  
)

TABLE TRACED (  
    NAME VARCHAR(32),  
    IPADR VARCHAR(39),  
    CHECKED TIMESTAMP NOT NULL,  
    MIN_RTT DOUBLE NOT NULL,  
    AVG_RTT DOUBLE,  
    C1 DOUBLE,  
    EPSILON DOUBLE,  
    HASH VARCHAR(64),  
    PRIMARY KEY(IPADR)  
)

TABLE MISC(  
    NAME VARCHAR(32),  
    IPADR VARCHAR(39),  
    LAST_BESTLINE TIMESTAMP,  
    BESTLINE_M DOUBLE,  
    BESTLINE_B DOUBLE,  
    LATITUDE DOUBLE,  
    LONGITUDE DOUBLE  
)
Appendix C

Source Code

The source code of the classes implementing the geolocation functionality in MAFIF along with management scripts and properties files are listed in this appendix. Classes belonging to MAFIF but not developed as part of the geolocation functionality are not included. The source code for these classes are available in [1].

C.1 AdminAgent Classes

C.1.1 AdminReplyGWBehaviour

```java
package kripos.geo;

import jade.content.Concept;
import jade.content.ContentElement;
import jade.content.lang.Codec.CodecException;
import jade.content.onto.OntologyException;
import jade.content.onto.UngroundedException;
import jade.content.onto.basic.Action;
import jade.core.AID;
import jade.core.behaviours.OneShotBehaviour;
import jade.lang.acl.ACLMessage;
import kripos.ontology.LogRec;
import kripos.ontology.OntDate;
import kripos.ontology.TraceResultList;

/**
 * @author oysteine
 * @version 1.0
 */
public class AdminReplyGWBehaviour extends OneShotBehaviour {
```
private static final long serialVersionUID = 
4025250051872250273L;

private AdminAgent myAdminAgent;
private ACLMessage myMsg;
private TraceResultList myTrl;

/**
 * @param a
 */
public AdminReplyGWBehaviour(AdminAgent a, ACLMessage msg) {
    super(a);
    myAdminAgent = a;
    myMsg = msg;
}

/**
 * Closes the case, logs appropriate information.
 * @see jade.core.behaviours.Behaviour#action()
 */
@Override
public void action() {
    try {
        AID aid = myMsg.getSender();
        ContentElement ce = myAdminAgent.getContentManager().
            extractContent(myMsg);
        Concept action = ((Action)ce).getAction();
        myTrl = (TraceResultList)action;
        String status = myTrl.getDoneStatus();

        if (status.equalsIgnoreCase("OK")){
            int cID = myAdminAgent.removeSessionAgentPackage(aid);
            myAdminAgent.removeActiveCase(cID);
            ACLMessage reply = myMsg.createReply();
            reply.setPerformative(ACLMessage.INFORM);
            myAgent.send(reply);

            /* Logs that the Case has been closed*/
            log("Case cID={0}SessionAgent terminated. Case end
                status: {1}" , myTrl.getDoneStatus(), "Type:Session", cID);
        //send die to SessionAgent

        //send data to GWAgent
        GWCase gwCase = myAdminAgent.getGWCase(cID);
        AID gwAID = gwCase.getAID();
        String convID = gwCase.getConversationID();

        //use myMsg for sending to GWAgent, already has correct
        //content, only set envelopeinfo
        myMsg.clearAllReceiver();
        myMsg.clearAllReplyTo();
        myMsg.addReceiver(gwAID);
    }
myMsg.setConversationId(convID);
myMsg.setSender(myAdminAgent.getAID());
myAgent.send(myMsg);
}
else if(status.equalsIgnoreCase("Fail")){
    int cID = myAdminAgent.removeSessionAgentPackage(aid);
    myAdminAgent.removeActiveCase(cID);
    //FIXME logging
    /*
     * log("Case closed. SessionAgent terminated.
     *     Case end status: " +
     *     myTrl.getDoneStatus() + "\n" + "Case
     *     FAILED due to: ")
     *     + myTrl.getHasReason(), "Type:Session",
     *     cID);
     */
    //send data to GWAgent
    ACLMessage reply = myMsg.createReply();
    reply.setPerformative(ACLMessage.NOT_UNDERSTOOD);
    myAgent.send(reply);
}
}
} catch (UngroundedException e) {
    e.printStackTrace();
} catch (CodecException e) {
    e.printStackTrace();
} catch (OntologyException e) {
    e.printStackTrace();
}
} ///action()

/**
 * Utility method for constructing LogRecs.
 * *
 * @param logContent The content of the LogRec
 * @param logType The Type of the LogRec
 * @param caseID The CaseID of the LogRec
 */
private void log(String logContent, String logType, int caseID){
    LogRec logr = new LogRec();
    logr.setColumnsCaseID(caseID);
    //logr.setHasSessionID(myCase.getCaseSessions().getCaseID());
    logr.setHasCaseID(myCase.getCaseSessions().getSessionID());
    logr.setHasLogType(logType);
    OntDate od = new OntDate();
    od.setSession(myAdminAgent.getCalendar().getSession());
    logr.setHasSessionID(od);
    logr.setHasLogType(logType);
    logr.setHasLogContent(logContent);
    myAdminAgent.getLogger().log(logr);
}
} //class
C.1.2  GWCase

```java
package kripos.geo;

import jade.core.AID;

/**
 * Utility class that bundles AID and Case information.
 * For use by AdminAgents to keep track of which GWAgent
 * requested which Case
 */

public class GWCase {
    private AID myAID;
    private int mySessionID;
    private int myCaseID;
    private String myConversationID;

    /**
     * Constructs an instance of GWCase
     * @param aid the AID of an agent
     * @param cID the case ID of a Case
     */
    public GWCase(AID aid, int cID, String convID) {
        super();
        myAID = aid;
        myCaseID = cID;
        myConversationID = convID;
    }

    /**
     * @return the myAID
     */
    public AID getAID() {
        return myAID;
    }

    /**
     * @return the myCaseID
     */
    public int getCaseID() {
        return myCaseID;
    }

    /**
     * @return the myConversationID
     */
    public String getConversationID() {
        return myConversationID;
    }
}
```

125
52
53 } //class
C.2 SessionAgent Classes

C.2.1 InitiateTraceSessionBehaviour

```java
package kripos.geo;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.Random;
import jade.content.ContentElement;
import jade.content.lang.Codec.CodecException;
import jade.content.lang.sl.SLCodec;
import jade.content.onto.OntologyException;
import jade.content.onto.UngroundedException;
import jade.content.onto.basic.Action;
import jade.content.onto.basic.Result;
import jade.core.AID;
import jade.core.ContainerID;
import jade.core.Location;
import jade.core.behaviours.OneShotBehaviour;
import jade.domain.JADEAgentManagement.CreateAgent;
import jade.domain.JADEAgentManagement.
    QueryPlatformLocationsAction;
import jade.domain.JADEAgentManagement.WhereIsAgentAction;
import jade.domain.mobility.MobilityOntology;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import kripos.gateway.CommandPackage;
import kripos.ontology.*;

/**
 * Behaviour that initiates a trace
 * @author oysteine
 * @version 1.0
 */
public class InitiateTraceSessionBehaviour extends OneShotBehaviour {
    private static final long serialVersionUID = -5048721800518980650L;
    //private ArrayList<Location> locations = new ArrayList<
    //Location>();
    private SessionAgent mySessionAgent;
    private Session mySession;
    private boolean disableLookupDB = true;
    private boolean disablePassive = true;
    /**
     * Creates an instance of this behaviour.
     */
```
* @param a The agent this instance belongs to.
*/

public InitiateTraceSessionBehaviour(SessionAgent a, Session s) {
    super(a);
    mySessionAgent = a;
    mySession = s;
}

/**
* Does initial work based on the type of trace session.
* @see jade.core.behaviours.OneShotBehaviour#action()
*/

@Override
public void action(){
    OntAddress toTrace = mySession.getHasBaseAddress();
    String params = mySession.getSessionParameters();
    // parse and check if dblookup
    // parse and check if passive
    // parse and check random-period for when to ping at a

    String sessionType = mySession.getSessionType();
    //fill commandpackage in owning SessionAgent
    CommandPackage atr = mySessionAgent.getAccTraceResults();
    if(toTrace instanceof IP) {
        atr.setTarget(((IP)toTrace).getHasName());
    }
    else if(toTrace instanceof DNSname){
        atr.setTarget(((DNSname)toTrace).getHasName());
    }
    atr.setType(sessionType);

    //TODO SEND 3 PINGS TO CHECK IF REACHABLE AT ALL BEFORE WASTING RESOURCES
    //TODO db lookup
    if(!disableLookupDB){
        dbLookup(toTrace);
    }

    if(!disablePassive){
        //TODO must return list of locations to use for trace
        ArrayList locations = passiveLookup(toTrace);
        //TODO superlandmark from this container?
        createWorkers(locations);
    }
    else {
        //TODO superlandmark from this container?
        createWorkers(null);
    }
// /* Logs the start of the Session*/ TOOD log correct information
// log("Session started. WorkerAgents created" + "\n" +
// SessionType: " + mySession.getSessionType() +
// "\n" + "Target: "+ mySession.getHasProtocol().getHasPrefix
// toString(),
// mySession.getHasBaseAddress()+mySession.getHasPath().
// toString(),
// "Type:Session", mySession.getCaseID());

private boolean dbLookup(OntAddress adr){ //not return true?
    boolean recent = false;
    //les inn terskel fra properties?
    //dboppslag
    //if(){} break and return traceresult directly from db.
    //else{
    return recent;
    // }
}

private ArrayList passiveLookup(OntAddress adr){
    ArrayList list = new ArrayList();
    return list;
}

/**
 * Creates the WorkerAgents that will perform the rest of
 * the delay measurements of the trace operation.
 * @param onLocations List of Locations to do measurements
 * from.
 * @return
 */
private void createWorkers(ArrayList onLocations){
    ArrayList<Location> locations = new ArrayList<Location>();
    AID myAMS = myAgent.getAMS();
}
// query AMS for platform available locations
QueryPlatformLocationsAction qpla = new QueryPlatformLocationsAction();
Action actAll = new Action(myAMS, qpla);
String convIDAll = sendRequest(actAll);

// receive locations
Result resAll = receiveInform(convIDAll);
jade.util.leap.Iterator it = resAll.getItems().iterator();

// uses all available locations
if (onLocations == null) {
    // due to unclear JadeGateway.init() documentation, need to filter out extra containers at mainhost
    ArrayList<String> onlyInclude = readLandmarks(connectDB());

    while (it.hasNext()) {
        Location loc = (Location)it.next();
        for (int i = 0; i < onlyInclude.size(); i++) {
            if (loc.getName().equalsIgnoreCase(onlyInclude.get(i))) {
                locations.add(loc);
                break;
            }
        }
    }

    // add own location
    WhereIsAgentAction wia = new WhereIsAgentAction();
    Action actSelf = new Action(myAMS, wia);
    String convSelf = sendRequest(actSelf);
    Result resSelf = receiveInform(convSelf);
    ContainerID cid = (ContainerID) resSelf.getValue();

    locations.add(cid);
}

// uses only supplied locations
else {
    while (it.hasNext()) {
        Location loc = (Location) it.next();
        for (int i = 0; i < onLocations.size(); i++) {
            // should optimize by removing from onLocation if match
            if (loc.getName().equalsIgnoreCase((String) onLocations.get(i))) {
                locations.add(loc);
            }
        }
    }

    // actually create agents
    for (int i = 0; i < locations.size(); i++) {
CreateAgent ca = new CreateAgent();
ca.setAgentName("Worker" + mySession.getSessionID() + String.valueOf(i));
ca.setClassName("kripos.geo.WorkerAgent");
ContainerID cid = (ContainerID)locations.get(i);
ca.setContainer(cid);
ca.addArguments(myAgent.getName());

WorkItem wi = new WorkItem();
wi.setCaseID(mySession.getCaseID());
wi.setSessionID(mySession.getSessionID());
wi.setHasBaseAddress(mySession.getHasBaseAddress());
wi.setSessionType(mySession.getSessionType());
ca.addArguments(wi);

Action a2 = new Action(myAMS, ca);
String convID2 = sendRequest(a2);

MessageTemplate mt = MessageTemplate.MatchConversationId(convID2);
ACLMessage resp = myAgent.blockingReceive(mt, 20000);
mySessionAgent.getConvList().deregisterConversation(resp.getConversationId());
}

locations.clear();
}

/**
 * Utility method for sending ACLMessages
 * <p>
 * Registers the conversationID in the convList
 * to avoid loosing any reply to the generic receivebehaviour
 * <p>
 * @param action the action that is to be wrapped in an ACLMessage
 * @return */
private String sendRequest(Action action){
 try {
 ACLMessage qMsg = new ACLMessage(ACLMessage.REQUEST);
 qMsg.setConversationId(myAgent.getName() + new Random().nextLong());
 qMsg.setLanguage(new SLCodec(0).getName());
 qMsg.setOntology(MobilityOntology.getInstance().getName());
 myAgent.getContentManager().fillContent(qMsg, action);
 qMsg.addReceiver(action.getActor());
 //register conversation with agent to get correct reception
 mySessionAgent.getConvList().registerConversation(qMsg.getConversationId());
 myAgent.send(qMsg);
return qMsg.getConversationId();

} catch (CodecException e) {
  e.printStackTrace();
  return null;
}

} catch (OntologyException e) {
  e.printStackTrace();
  return null;
}

} // **

/**
* Utility method for receiving ACLMessages with matching conversation IDs
* @param cid the conversation id to match
* @return the content of the ACLMessage
*/
private Result receiveInform(String cid){
  try {
    MessageTemplate mt = MessageTemplate.MatchConversationId(cid);
    ACLMessage resp = myAgent.blockingReceive(mt, 20000);
    ContentElement ce = myAgent.getContentManager().
      extractContent(resp);
    Result result = (Result) ce;
    mySessionAgent.getConvList().deregisterConversation(cid);
    return result;
  }

} catch (UngroundedException e) {
  e.printStackTrace();
  mySessionAgent.getConvList().deregisterConversation(cid);
  return null;
}

} catch (CodecException e) {
  e.printStackTrace();
  mySessionAgent.getConvList().deregisterConversation(cid);
  return null;
}

} catch (OntologyException e) {
  e.printStackTrace();
  mySessionAgent.getConvList().deregisterConversation(cid);
  return null;
}

}

} // **

/**
* Reads landmarks from the local database.
* @param con Database connection
* @return a list of landmarks
*/
private ArrayList<String> readLandmarks(Connection con){
  ArrayList<String> landmarkNames = new ArrayList<String>();
  }
try {
    Statement stmt = con.createStatement();
    String query = "SELECT \_NAME\_ FROM \_LANDMARKS\_";
    ResultSet rs = stmt.executeQuery(query);
    while (rs.next()){
        String name = rs.getString("NAME");
        landmarkNames.add(name);
    }
} catch (SQLException e) {
    // TODO: handle exception
}
return landmarkNames;

/**
 * Connects to the local database and returns the Connection
 * for further use.
 *
 * @return a connection to the database
 */
private Connection connectDB(){
    try{
        Class.forName("org.hsqldb.jdbcDriver");
        c = DriverManager.getConnection("jdbc:hsqldb:hsql://localhost/xdb", "sa", "");
    } catch (ClassNotFoundException e) {
        //FIXME unable to find database classes. dosomething
        e.printStackTrace();
    } catch (SQLException e) {
        //FIXME unable to connect to database dosomething
        e.printStackTrace();
    }
    return c;
}

/**
 * Utility method for constructing LogRecs and logging them.
 *
 * @param logContent The content of the LogRec
 * @param logType The Type of the LogRec
 * @param caseID The CaseID of the LogRec
 */
private void log(String logContent, String logType, int caseID){
    LogRec logr = new LogRec();
    logr.setHasCaseID(caseID);
    logr.setHasSessionID(mySession.getSessionID());
    logr.setLogType(logType);
    OntDate od = new OntDate();
    od.setTime(mySessionAgent.getCalendar().getTimeInMillis());
    logr.setHasDate(od);
}
logr.setLogLevel("INFO");
logr.setLogContent(logContent);
mySessionAgent.getLogger().log(logr);
} //class
C.2.2 TraceReduceBehaviour

```java
package kripos.geo;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.Iterator;
import jade.content.ContentElement;
import jade.content.lang.Codec.CodecException;
import jade.content.lang.sl.SLCodec;
import jade.content.onto.OntologyException;
import jade.content.onto.UngroundedException;
import jade.content.onto.basic.Action;
import jade.core.AID;
import jade.core.behaviours.OneShotBehaviour;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import kripos.gateway.CommandPackage;
import kripos.ontology.*;

public class TraceReduceBehaviour extends OneShotBehaviour {

    // Constructor
    public TraceReduceBehaviour(SessionAgent a, ACLMessage m) {
        super(a);
        mySessionAgent = a;
        myMsg = m;
    }

    // Class variables
    private static final long serialVersionUID = -7988728778910969955L;
    private SessionAgent mySessionAgent;
    private ACLMessage myMsg;
    private AID myWorkerAID;
    private IP myTarget;
    private String myType = null;
    private String myWorkerDoneStatus = "OK";

    // Method
    public TraceReduceBehaviour(SessionAgent a, ACLMessage m) {
        super(a);
        mySessionAgent = a;
        myMsg = m;
    }
```
/** Utility method for constructing LogRecs and logging them. *
 * @param logContent The content of the LogRec
 * @param logType The Type of the LogRec
 * @param caseID The CaseID of the LogRec
 */

private void log(String logContent, String logType, int caseID)
{
    LogRec logr = new LogRec();
    logr.setHasCaseID(caseID);
    logr.setHasSessionID(mySessionAgent.getSession().getSessionID());
    logr.setLogType(logType);
    OntDate od = new OntDate();
    od.setTime(mySessionAgent.getCalendar().getTimeInMillis());
    logr.setHasDate(od);
    logr.setLogLevel("INFO");
    logr.setLogContent(logContent);
    mySessionAgent.getLogger().log(logr);
}

/* (non-Javadoc)
 * @see jade.core.behaviours.Behaviour#action()
 */

@Override
public void action() {
    // TODO update db?
    try {
        myWorkerAID = myMsg.getSender();
        ContentElement ce = mySessionAgent.getContentManager().extractContent(myMsg);
        TraceResultList trl = (TraceResultList)((Action)ce).getAction();
        ArrayList<TraceResult> traceResults = new ArrayList<TraceResult>();
        Iterator it = trl.getAllTraceResult();
        while(it.hasNext()){
            traceResults.add((TraceResult)it.next());
        }
        myTarget = (IP)(traceResults.get(0)).getAddressToBeTraced();
        if(traceResults.size()<2){//TODO check type instead!!
            //TODO check for error
            myType = "TRACE-CBG";
            trl.getDoneStatus();//TODO set in total and discard
            (mySessionAgent.getAccTraceResults()).addResult(
                traceResults.get(0));
        } else {
            myType = "TRACE-GeoPing";
            ArrayList<DelayVector> dVectors = mySessionAgent.getAccTraceResults().getDelayVectors();
        }
    }
if (dVectors.isEmpty()) {
    ArrayList<String> names = landmarkNames(myTarget);
    for (int i = 0; i < names.size(); i++) {
        DelayVector dv = new DelayVector(names.get(i));
        dVectors.add(dv);
    }
    for (int i = 0; i < dVectors.size(); i++) {
        for (int j = 1; j < traceResults.size(); j++) {
            // skip the TraceResult to target
            if (dVectors.get(i).getName().equalsIgnoreCase((
                DNSname)(traceResults.get(j).getAddressToBeTraced()).getHasName())) {
                dVectors.get(i).add(traceResults.remove(j).getTraceResultData(),
                traceResults.get(0).getTraceResultData());
            }
        }
    }
    // TODO myWorkerDoneStatus = done.getDoneStatus(); // fix tilsvarende i trace
    // TODO mySessionAgent.setDoneStatus(myWorkerDoneStatus);
    if (myWorkerDoneStatus.equalsIgnoreCase("OK")) {
        reduce();
        log("WorkerAgent:" + myWorkerAID + "finished, successfully, terminated", "Type:Session",
        mySessionAgent.getSession().getCaseID());
    } else if (myWorkerDoneStatus.equalsIgnoreCase("Fail")) {
        // Should do something more. Why failed? Create new worker?
        reduce();
        log("WorkerAgent:" + myWorkerAID + "failed, due to " + // FIXME done.getHasReason() + "",
        "terminated", "Type:Session",
        mySessionAgent.getSession().getCaseID());
    } else {
        ACLMessage reply = myMsg.createReply();
        reply.setPerformative(ACLMessage.NOT_UNDERSTOOD);
        myAgent.send(reply);
    }
}
}
} catch (UngroundedException e) {
    e.printStackTrace();
} catch (CodecException e) {
    e.printStackTrace();
} catch (OntologyException e) {
    e.printStackTrace();
}
e.printStackTrace();
}
} /*action()
/**
* Cleans up after a WorkerAgent is Done, and optionally terminates the entire Session.
* <p>
* Removes the workerAgent that sent the TraceList from myWorkerAgents.
* Sets the doneStatus of the sessionAgent to the doneStatus in the incoming Done.
* Checks if there are any more WorkerAgents, if not terminate the session and kill
* the owning SessionAgent after informing its AdminAgent.
*/
private void reduce(){
    mySessionAgent.removeWorker(myWorkerAID);
    //TODO mySessionAgent.setDoneStatus(myWorkerDoneStatus);
    if (mySessionAgent.getWorkers().isEmpty()){
        TraceResultList trl = new TraceResultList();
        CommandPackage cp = mySessionAgent.getAccTraceResults();
        if(myType.equalsIgnoreCase("Trace-CBG")){
            ArrayList<TraceResult> results = cp.getCBGResults();
            for(int i=0;i<results.size();i++){
                trl.addTraceResult(results.get(i));
            }
        }
        else if(myType.equalsIgnoreCase("Trace-GeoPing")){
            ArrayList<DelayVector> dList = cp.getDelayVectors();
            ArrayList<Object[]> lInfo = landmarkInfo(myTarget);
            double one = 99999999; //arbitrary high values
            double two = 99999999;
            double three = 99999999;
            double four = 99999999;
            TraceResult tr1 = new TraceResult();
            TraceResult tr2 = new TraceResult();
            TraceResult tr3 = new TraceResult();
            TraceResult tr4 = new TraceResult();
            for (int i=0;i<dList.size();i++){
                Double temp = dList.get(i).euclidianDistance();
                if(temp<one && temp>0){
                    one = temp;
                    for(int j=0;j<lInfo.size();j++){
                        if(dList.get(i).getName().equalsIgnoreCase(((String)lInfo.get(j)[0])){
                            GeoLocation geo = new GeoLocation();
                            geo.setLocationName(((String)lInfo.get(j)[0]));
                            geo.setLocationLatitude(((Double)lInfo.get(j) [1]).floatValue());
                        }
                    }
                }
            }
        }
    }
}
geo.setLocationLongitude(((Double)lInfo.get(j)[2]).floatValue());
tr1.setHasGeoLocation(geo);
}
}

tr1.setTraceResultData(temp.floatValue());
else if(temp<two && temp>0){
two = temp;
for(int j=0;j<lInfo.size();j++){
    if(dList.get(i).getName().equalsIgnoreCase((String)lInfo.get(j)[0])){
        GeoLocation geo = new GeoLocation();
        geo.setLocationName((String)lInfo.get(j)[0]);
        geo.setLocationLatitude(((Double)lInfo.get(j)[1]).floatValue());
        geo.setLocationLongitude(((Double)lInfo.get(j)[2]).floatValue());
        tr2.setHasGeoLocation(geo);
    }
}
tr2.setTraceResultData(temp.floatValue());
}
else if(temp<three && temp>0){
    three = temp;
    for(int j=0;j<lInfo.size();j++){
        if(dList.get(i).getName().equalsIgnoreCase((String)lInfo.get(j)[0])){
            GeoLocation geo = new GeoLocation();
            geo.setLocationName((String)lInfo.get(j)[0]);
            geo.setLocationLatitude(((Double)lInfo.get(j)[1]).floatValue());
            geo.setLocationLongitude(((Double)lInfo.get(j)[2]).floatValue());
            tr3.setHasGeoLocation(geo);
        }
    }
    tr3.setTraceResultData(temp.floatValue());
}
else if(temp<four && temp>0){
    four = temp;
    for(int j=0;j<lInfo.size();j++){
        if(dList.get(i).getName().equalsIgnoreCase((String)lInfo.get(j)[0])){
            GeoLocation geo = new GeoLocation();
            geo.setLocationName((String)lInfo.get(j)[0]);
            geo.setLocationLatitude(((Double)lInfo.get(j)[1]).floatValue());
            geo.setLocationLongitude(((Double)lInfo.get(j)[2]).floatValue());
            tr4.setHasGeoLocation(geo);
        }
    }
tr4.setTraceResultData(temp.floatValue());

trl.addTraceResult(tr1);
trl.addTraceResult(tr2);
trl.addTraceResult(tr3);
trl.addTraceResult(tr4);

trl.setDoneStatus(mySessionAgent.getDoneStatus());
ACLMessage isDone = new ACLMessage(ACLMessage.INFORM);
isDone.addReceiver(mySessionAgent.getAdmin());
isDone.setLanguage(new SLCodec(0).getName());
isDone.setOntology(InternetInvestigationsOntology.getInstance().getName());
String convID = mySessionAgent.getConvList().registerConversation();
isDone.setConversationId(convID);
Action act = new Action(mySessionAgent.getAdmin(), trl);

try {
    mySessionAgent.getContentManager().fillContent(isDone, act);
    mySessionAgent.send(isDone);
} catch (CodecException e) {
    e.printStackTrace();
} catch (OntologyException e) {
    e.printStackTrace();
}

ACLMessage doneReply = mySessionAgent.blockingReceive(
    MessageTemplate.MatchConversationId(convID), 90000);
if (doneReply != null){
    if (doneReply.getPerformative() == ACLMessage.INFORM);
        /* Log that we have reported done and terminated after receiving reply*/
        log(myAgent.getName() + "finished", "Type:Session",
            mySessionAgent.getSession().getCaseID());
    myAgent.doDelete();
} else{
    /* Log that we have reported done and terminated on timeout*/
    log(myAgent.getName() + "finished" + "selfterminated",
        "Type:Session", mySessionAgent.getSession().
            getCaseID());
    myAgent.doDelete();
} //outer if
//reduce()

/**
 * Get the names of landmarks from the Landmarks database,
 * excluding any landmark matching the input IP adress.
 *
 * @param IP address to exclude
 * @return the number of landmarks
 */
private ArrayList<String> landmarkNames(IP ip){
    Connection con = connectDB();
    ArrayList<String> names = new ArrayList<String>();
    try {
        Statement stmt = con.createStatement();
        String query = "SELECT NAME FROM LANDMARKS WHERE IPADR <> '" + ip.getHasName() + "'";
        ResultSet rs = stmt.executeQuery(query);
        while (rs.next()){
            names.add(rs.getString(1));
        }
    } catch (SQLException se){
        se.printStackTrace();
        return names;
    } return names;
}

/**
 * Get the names and geographic locations of landmarks from
 * the Landmarks database,
 * excluding any landmark matching the input IP adress.
 *
 * @param IP address to exclude
 * @return an array containing the name and latitude and
 * longitude
 */
private ArrayList<Object[]> landmarkInfo(IP ip){
    Connection con = connectDB();
    ArrayList<Object[]> info = new ArrayList<Object[]>();
    try {
        Statement stmt = con.createStatement();
        String query = "SELECT NAME, LATITUDE, LONGITUDE FROM LANDMARKS WHERE IPADR <> '" + ip.getHasName() + "'";
        ResultSet rs = stmt.executeQuery(query);
        while (rs.next()){
            Object[] element = new Object[3];
            element[0] = rs.getString("NAME");
            element[1] = new Double(rs.getDouble("LATITUDE"));
            element[2] = new Double(rs.getDouble("LONGITUDE"));
            info.add(element);
        }
    } catch (SQLException se){
        //log error
    }
se.printStackTrace();
return info;
}
return info;

/**
 * Connects to the local database and returns the Connection for further use.
 *
 * @return a connection to the database
 *
 */
private Connection connectDB(){
    Connection c = null;
    try {
        Class.forName("org.hsqldb.jdbcDriver");
        c = DriverManager.getConnection("jdbc:hsqldb:hsql://localhost/xdb", "sa", "");
    } catch (ClassNotFoundException e) {
        //FIXME unable to find database classes. dosomething
        e.printStackTrace();
    } catch (SQLException e) {
        //FIXME unable to connect to database dosomething
        e.printStackTrace();
    }
    return c;
}
//class
C.2.3 DelayVector

```java
package kripos.geo;

import java.util.ArrayList;

/**
 * A delay vector for a given landmark.
 * @author oysteine
 * @version 1.0
 */
public class DelayVector extends ArrayList {
    private static final long serialVersionUID = 7614366734429571963L;
    private String myName;
    private double sum;

    /**
     * @param name The name of the landmark represented by this instance
     */
    public DelayVector(String name) {
        myName = name;
    }

    /**
     * Adds a delay component to this DelayVector
     * @param toThis the delay from calling landmark to this landmark
     * @param toTarget the delay from calling landmark to the target
     */
    public void add(double toThis, double toTarget){
        Double temp = new Double((toThis-toTarget)*(toThis-toTarget));
        this.add(temp);
        sum = sum+temp;
    }

    /**
     * Computes the Euclidian distance of this DelayVector
     * @return the euclidian distance
     */
    public double euclidianDistance(){
        if (sum>0){
            return Math.sqrt(sum);
        }
        else {
            return -1;
        }
    }
}
```
public String getName() {
    return myName;
}

} // class
C.3 WorkerAgent Classes

C.3.1 WorkerTraceStart

```java
package kripos.geo;

import java.net.InetAddress;
import java.net.UnknownHostException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.Date;
import com.bbn.openmap.LatLonPoint;
import jade.core.behaviours.OneShotBehaviour;
import jade.core.behaviours.SequentialBehaviour;
import kripos.ontology.DNSname;
import kripos.ontology.IP;
import kripos.ontology.OntAddress;
import kripos.ontology.WorkItem;

public class WorkerTraceStart extends OneShotBehaviour {
    private static final long serialVersionUID = 82443708805797669L;
    private WorkerAgent myWorkerAgent;
    private WorkItem myWorkItem;
    private long allowedSlack = 0; // TODO set sane value

    public WorkerTraceStart(WorkerAgent a, WorkItem wi) {
        super(a);
        myWorkerAgent = a;
        myWorkItem = wi;
    }

    @Override
    public void action() {
        ArrayList<Landmark> landmarks = new ArrayList<Landmark>();
        double[] bestLine = new double[2];
        Landmark toTrace = null;
        try {
            toTrace = new Landmark(lookupIP().get(0)); // TODO only uses first returned IP
        } catch (UnknownHostException e1) {
            e1.printStackTrace();
        }
        Connection con = connectDB();
        ArrayList<Landmark> initialLandmarks = readLandmarks(con);
        if (checkOldBestline(con)) {
            landmarks = initialLandmarks;
        }
    }
}
```
try {
    Statement stmt = con.createStatement();
    String query = "SELECT BESTLINE_M, BESTLINE_B FROM MISCF"
    ;
    ResultSet rs = stmt.executeQuery(query);
    while (rs.next()){
        bestLine[0] = rs.getDouble("BESTLINE_M");
        bestLine[1] = rs.getDouble("BESTLINE_B");
    }

    ArrayList<String> ips = newlyTraced(con);
    if (ips.get(ips.size()-1).equalsIgnoreCase("true")){
        toTrace = new Landmark(ips.get(0));
        try {
            Statement st = con.createStatement();
            String query2 = "SELECT NAME, MIN_RTT, C1, EPSILON,
            IPADR, CHECKED FROM TRACED WHERE IPADR = "+ips
            .get(0)++";+
            ResultSet rs2 = st.executeQuery(query2);
            while (rs2.next()){
                double minRtt = rs2.getDouble("MIN_RTT");
                double C1 = rs2.getDouble("C1");
                double epsilon = rs2.getDouble("EPSILON");
                toTrace.setMinRTT(minRtt);
                toTrace.setC1(C1);
                toTrace.setEpsilon(epsilon);
            }
            ((SequentialBehaviour)root()).addSubBehaviour(
                new WorkerTraceFinal(myWorkerAgent, toTrace, bestLine, landmarks));
        }
        catch (SQLException e) {
            e.printStackTrace();
        }
        }
    }
    else {
        ((SequentialBehaviour)root()).addSubBehaviour(
            new WorkerTraceTarget(myWorkerAgent, toTrace, bestLine, landmarks));
    }
    }
    catch(SQLException se){
        //FIXME do something
        se.printStackTrace();
    }
    }
    }
else{
    WorkerTraceLandmarks part2 = new WorkerTraceLandmarks(
        myWorkerAgent, toTrace);
    myWorkerAgent.getPingList().setContent(initialLandmarks);
    for (int i=0; i<initialLandmarks.size(); i++){
        PingBehave pb = new PingBehave(myWorkerAgent, initialLandmarks.get(i));
        part2.addSubBehaviour(part2.getTbf().wrap(pb));
    }
}
((SequentialBehaviour)root()).addSubBehaviour(part2);
}
//action
/**
 * If the class of myTarget is DNSName we need to get the IP
 * address.
 * If the class of myTarget is IP no lookup is performed
 * before returning the IP address.
 *
 * @return An ArrayList of IP addresses as strings
 * @throws UnknownHostException
 */
private ArrayList<String> lookupIP() throws UnknownHostException {
    ArrayList<String> ipStrings = null;
    OntAddress oa = myWorkItem.getHasBaseAddress();
    if (oa instanceof DNSname) {
        DNSName dns = (DNSName)(oa);
        InetAddress[] ips;
        ips = InetAddress.getAllByName(dns.getHasName());
        ipStrings = new ArrayList<String>();
        for(int i=0;i<ips.length;i++){
            ipStrings.add(ips[i].getHostAddress());
        }
        return ipStrings;
    } else {
        ipStrings = new ArrayList<String>();
        ipStrings.add(((IP)myWorkItem.getHasBaseAddress()).
                      getHasName());
        return ipStrings;
    }
}

/**
 * Reads landmarks from the local database.
 *
 * @param con Database connection
 * @return a list of landmarks sorted by increasing distance
 *         from the owner of this instance
 */
private ArrayList<Landmark> readLandmarks(Connection con){
    ArrayList<Landmark> landmarks = new ArrayList<Landmark>();
    try {
        Statement stmt = con.createStatement();
        String query = "SELECT * FROM LANDMARKS ORDER BY "
                       "DISTANCE_KM ASC";
        ResultSet rs = stmt.executeQuery(query);
        while(rs.next()){

String name = rs.getString("NAME");
String ipAdr = rs.getString("IPADR");
double distance = rs.getDouble("DISTANCE_KM");
LatLonPoint llp = new LatLonPoint(rs.getDouble("LATITUDE"), rs.getDouble("LONGITUDE"));

Landmark l = new Landmark(name, llp, ipAdr);
l.setDistance(distance);
landmarks.add(l);
}
}
}
}
try {
ArrayList<String> adrToCheck = lookupIP();
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
for (int i=0;i<adrToCheck.size();i++){
String query = "SELECT NAME, IPADR, CHECKED FROM TRACED WHERE IPADR = '"+adrToCheck.get(i)+"';";
ResultSet rs = stmt.executeQuery(query);
if (!rs.next()){
returnList.add(adrToCheck.get(i));
}
rs.previous();
while (rs.next()) {
Date lastChecked = (Date)rs.getTimestamp("CHECKED");
long toCheck;
if (rs.wasNull()){
toCheck = 0;
}
else{
toCheck = lastChecked.getTime();
}
long currentTime = myWorkerAgent.getCalendar().getTimeInMillis();
long diff = currentTime - toCheck;
if (diff<allowedSlack){
returnList.add(rs.getString("IPADR"));
newlyTraced = "true";
}
193 }  
194 else{  
195    returnList.add(adrToCheck.get(i));  
196 }  
197 }  
198 }  
199 }  
200 catch (UnknownHostException uhe){  
201     uhe.printStackTrace();  
202     returnList.add("false");  
203     return returnList;  
204 }  
205 catch (SQLException e) {  
206     e.printStackTrace();  
207     returnList.add(newlyTraced);  
208     return returnList;  
209 }  
210 returnList.add(newlyTraced);  
211 return returnList;  
212 }  
213  
214 /**  
215 * Check if the most recently calculated bestLine for this  
216 * host is too old.  
217 * @return true if the current baseLine is usable  
218 */  
219 private boolean checkOldBestline(Connection con){  
220     Date lastChecked = new Date(0);  
221     try {  
222         Statement stmt = con.createStatement();  
223         String query = "SELECT LAST_BESTLINE FROM MISC";  
224         ResultSet rs = stmt.executeQuery(query);  
225         while (rs.next()) {  
226             if(rs.getTimestamp("LAST_BESTLINE") != null){  
227                 lastChecked = (Date)rs.getTimestamp("LAST_BESTLINE");  
228             }  
229         }  
230     } catch (SQLException e) {  
231         e.printStackTrace();  
232         return false;  
233     }  
234     long toCheck = lastChecked.getTime();  
235     long currentTime = myWorkerAgent.getCalendar().getTimeInMillis();  
236     long diff = currentTime - toCheck;  
237     if(diff <= allowedSlack){  
238         return true;  
239     } else{
    return false;
}

/**
 * Connects to the local database and returns the Connection for further use.
 *
 * @return a connection to the database
 */
private Connection connectDB(){//TODO path & password
    Connection c = null;
    try {
        Class.forName("org.hsqldb.jdbcDriver");
        c = DriverManager.getConnection("jdbc:hsqldb:hsqldb://localhost/xdb", "sa", ");
    } catch (ClassNotFoundException e) {
        //FIXME unable to find database classes. do something
        e.printStackTrace();
    } catch (SQLException e) {
        //FIXME unable to connect to database do something
        e.printStackTrace();
    }
    return c;
} //class
C.3.2 WorkerTraceLandmarks

```java
package kripos.geo;

import jade.core.behaviours.ParallelBehaviour;
import jade.core.behaviours.SequentialBehaviour;
import jade.core.behaviours.ThreadedBehaviourFactory;

/**
 * Wrapper for multiple PingBehaves running in dedicated threads
 * @author oysteine
 */
public class WorkerTraceLandmarks extends ParallelBehaviour {
    private static final long serialVersionUID = -2880427229534140463L;
    private ThreadedBehaviourFactory myTbf;
    private Landmark myTarget;
    private WorkerAgent myWorkerAgent;

    public WorkerTraceLandmarks(WorkerAgent a, Landmark toTrace) {
        super(a, ParallelBehaviour.WHEN_ALL);
        myTbf = new ThreadedBehaviourFactory();
        myTarget = toTrace;
        myWorkerAgent = a;
    }

    /**
     * @return the <code>ThreadedBehaviourFactory</code> of this instance
     */
    public ThreadedBehaviourFactory getTbf() {
        return myTbf;
    }

    @Override
    public int onEnd() {
        ((SequentialBehaviour) root()).addSubBehaviour(new WorkerTraceCalculate(myWorkerAgent, myTarget));
        return 0;
    }
}
```
C.3.3 WorkerTraceCalculate

```java
package kripos.geo;

import java.net.InetAddress;
import java.net.UnknownHostException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.Date;
import Jama.Matrix;
import jade.core.behaviours.OneShotBehaviour;
import jade.core.behaviours.SequentialBehaviour;
import kripos.ontology.DNSname;
import kripos.ontology.IP;

public class WorkerTraceCalculate extends OneShotBehaviour {
    private WorkerAgent myWorkerAgent;
    private Landmark toTrace;
    private double[] myBestLine = new double[2];
    private ArrayList<Landmark> myLandmarks;
    private double baseM = 0.02; // magic number due to lightspeed in optical fibres
    private long allowedSlack = 0;

    public WorkerTraceCalculate(WorkerAgent a, Landmark toTrace) {
        super(a);
        myWorkerAgent = a;
        toTrace = toTrace;
    }

    @Override
    public void action() {
        myLandmarks = myWorkerAgent.getPingList().getLandmarkList();
        for (int o = 0; o < myLandmarks.size(); o++) {
            Landmark l = myLandmarks.get(o);
        }
        if (myWorkerAgent.getType().equalsIgnoreCase("TRACE-CBG")) {
            myBestLine = calculateBestline(myLandmarks);
        }
        Connection con = connectDB();
    }
}
```
updateDB(con, myBestLine, myLandmarks);

ArrayList<String> ips = newlyTraced(con);

if (ips.get(ips.size()-1).equalsIgnoreCase("true")){
    toTrace = new Landmark(ips.get(0));
    Statement st;
    try {
        st = con.createStatement();
        String query2 = "SELECT NAME, MIN_RTT, C1, EPSILON, IPADR, CHECKED FROM TRACED WHERE IPADR = '"+ips.get(0)+"'";
        ResultSet rs2 = st.executeQuery(query2);
        while (rs2.next()){
            double minRtt = rs2.getDouble("MIN_RTT");
            double c1 = rs2.getDouble("C1");
            double epsilon = rs2.getDouble("EPSILON");
            toTrace.setMinRTT(minRtt);
            toTrace.setC1(c1);
            toTrace.setEpsilon(epsilon);
        }
        ((SequentialBehaviour)root()).addSubBehaviour(
            new WorkerTraceFinal(myWorkerAgent, toTrace, myBestLine, myLandmarks));
    }catch (SQLException e) {
        e.printStackTrace();
    }
}
else{  
    ((SequentialBehaviour)root()).addSubBehaviour(
        new WorkerTraceTarget(myWorkerAgent, toTrace, myBestLine, myLandmarks));
}

/**
 * Calculates the bestline from ping measurements
 * @param landmarks the landmarks used as a basis for calculating the bestline
 * @return an array containing the two double values that constitutes the bestline
 */
private double[] calculateBestline(ArrayList<Landmark> landmarks){
    double[] result = new double[2];
    double bestMi = 9999999; //initial nonsense values
    double bestBi = 9999999;
    double smallestY = 9999999;
    double smallestX = 0;
    int startAt = 1;  //requires sorted input!
    for(int i=0;i<landmarks.size();i++){
double tempY = landmarks.get(i).getMinRTT();
// set new value for smallestY, but only if it is sane
if (tempY < smallestY && landmarks.get(i).getMinRTT() >=
    landmarks.get(i).getDistance() * baseM){
    smallestY = tempY;
    startAt = i+1; // only use landmarks with greater
distance for calculating bestline
    smallestX = landmarks.get(i).getDistance();
}

// set up matrices for solving y = mx + b
Matrix a = new Matrix(2,2);
Matrix b = new Matrix(2,1);
// fill matrices with values from landmark containing
smallestY
a.set(0, 0, smallestX);
a.set(0, 1, 1);
b.set(0, 0, smallestY);

// solve y = mx + b for all the pairs (smallestY, other
// landmark)
for(int i=startAt;i<landmarks.size();i++){  
a.set(1, 0, landmarks.get(i).getDistance());
a.set(1, 1, 1);
b.set(1, 0, landmarks.get(i).getMinRTT());

Matrix x = a.solve(b);
if((x.get(0,0)>baseM) && (x.get(1,0)>=0)){
    if(x.get(0,0)< bestMi){
        bestMi = x.get(0,0);
        bestBi = x.get(1,0);
    }
}
}

// choose next-smallest Y if result does not fall within
allowed region
if(bestMi<baseM || bestBi<0){
    if(landmarks.size()>4){
        landmarks.remove(startAt-1); // remove the outlier
        calculateBestline(landmarks);
    }
    result[0] = bestMi;
    result[1] = bestBi;
    return result; // TODO throw exception instead
}
else{
    result[0] = bestMi;
    result[1] = bestBi;
    return result;
}
}
/**
 * Connects to the local database and returns the Connection for further use.
 * @return a connection to the database
 */
private Connection connectDB(){
    Connection c = null;
    try {
        Class.forName("org.hsqldb.jdbcDriver");
        c = DriverManager.getConnection("jdbc:hsqldb:hsql://
localhost/xdb", "sa", "");
    } catch (ClassNotFoundException e) {
        //FIXME unable to find database classes. dosomething
        e.printStackTrace();
    } catch (SQLException e) {
        //FIXME unable to connect to database dosomething
        e.printStackTrace();
    }
    return c;
}

/**
 * Checks if an IP address has currently been traced.
 * The definition of current is given by allowedSlack.
 * @param con The database connection to use
 * @return true if this IP was recently traced
 */
private ArrayList<String> newlyTraced(Connection con){
    String newlyTraced = "false";
    ArrayList<String> returnList = new ArrayList<String>();
    try {
        String adrToCheck = toTrace.getIP();
        Statement stmt = con.createStatement(ResultSet.
                         TYPE_SCROLL_INSENSITIVE,ResultSet.CONCUR_READ_ONLY);
        String query = "SELECT NAME, IPADR, CHECKED FROM TRACED WHERE IPADR = " + adrToCheck + "";
        ResultSet rs = stmt.executeQuery(query);
        if (!rs.next()){
            returnList.add(adrToCheck);
        }
        rs.previous();
        while (rs.next()) {
            Date lastChecked = (Date)rs.getTimestamp("CHECKED");
            long toCheck;
            if(rs.wasNull()){
                toCheck = 0;
            }
            else{
                toCheck = lastChecked.getTime();
            }
        }
    }
    return returnList;
}
long currentTime = myWorkerAgent.getCalendar().get(Calendar.MILLISECOND);
long diff = currentTime - toCheck;
if (diff < allowedSlack) {
    returnList.add(rs.getString("IPADR"));
    newlyTraced = "true";
}
else {
    returnList.add(adrToCheck);
}
}
}
}
catch (SQLException e) {
    e.printStackTrace();
    returnList.add(newlyTraced);
    return returnList;
}
returnList.add(newlyTraced);
return returnList;
}
/**
** Updates the MISC table with the new bestline information and any matching landmarks in the LANDMARK database with new minRTT, C1, Epsilon and timestamp
**
** @param con The Connection to use
** @param mi the m part of the new bestline
** @param bi the b part of the new bestline
*/
private void updateDB(Connection con, double[] bestline, ArrayList<Landmark> newLandmarks) {
    String newBestline = "UPDATE MISC SET LAST_BESTLINE = now, BESTLINE_M = " + bestline[0] + " , BESTLINE_B = " + bestline[1];
    try {
        Statement st = con.createStatement();
        st.executeUpdate(newBestline);
        for (int i = 0; i < newLandmarks.size(); i++) {
            String landmarkUpdate = "UPDATE LANDMARKS SET MIN_RTT = " + newLandmarks.get(i).getMinRTT() + " , CHECKED = now , C1 = " + newLandmarks.get(i).getC1() + " , EPSILON = " + newLandmarks.get(i).getEpsilon();
            st.executeUpdate(landmarkUpdate);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
public class WorkerTraceTarget extends PingBehave {
    private static final long serialVersionUID = 1898594180371430543L;
    private WorkerAgent myWorkerAgent;
    private double[] myBestLine;
    private ArrayList<Landmark> myLandmarks;
    public WorkerTraceTarget(WorkerAgent a, Landmark target, double[] bestLine, ArrayList<Landmark> landmarks) {
        super(a, target);
        myWorkerAgent = a;
        myBestLine = bestLine;
        myLandmarks = landmarks;
    }
    public WorkerTraceTarget(WorkerAgent a, Landmark target, int numPings, int pingDistance, boolean estimateEps, boolean estimateC1, double[] bestLine, ArrayList<Landmark> landmarks) {
        super(a, target, numPings, pingDistance, estimateEps, estimateC1);
        myWorkerAgent = a;
        myBestLine = bestLine;
        myLandmarks = landmarks;
    }
    @Override
    public void action() {
        super.action();
        updateTracedDB(connectDB(), myTarget);
        ((SequentialBehaviour)root()).addSubBehaviour(new WorkerTraceFinal(myWorkerAgent, myTarget, myBestLine, myLandmarks));
    }
    /**
     * Creates a new entry in the TRACED database,
* or updates an existing entry if a matching IP address is found
* @param con The database connection to use
* @param toTrace the landmark containing the information to added
*
private void updateTracedDB(Connection con, Landmark toTrace)
{
    try {
        Statement stmt = con.createStatement();
        String query = "SELECT COUNT(IPADR) FROM TRACED WHERE IPADR='" + toTrace.getIP() + "';
        ResultSet rs = stmt.executeQuery(query);
        while (rs.next()){
            if(rs.getInt(1)>0){
                String updateString = "UPDATE TRACED SET MIN_RTT=" + toTrace.getMinRTT() +", CHECKED=now WHERE IPADR='" + toTrace.getIP() + "';
                stmt.executeUpdate(updateString);
            }
        }
    }
    else{
        String ip = toTrace.getIP();
        String name = toTrace.getName();
        double c1 = toTrace.getC1();
        double epsilon = toTrace.getEpsilon();
        double minRtt = toTrace.getMinRTT();
        String insertString = "INSERT INTO TRACED VALUES ('" + name + ","," + ip + ",now," + minRtt + ",null," + c1 + "," + epsilon + ",null');
        stmt.executeUpdate(insertString);
    }
}
} catch (SQLException e) {
    e.printStackTrace();
}
/**
 * Connects to the local database and returns the Connection for further use.
 * @return a connection to the database
 */
private Connection connectDB(){
    Connection c = null;
    try {
        Class.forName("org.hsqldb.jdbcDriver");
        c = DriverManager.getConnection("jdbc:hsqldb:hsq1://localhost/xdb", "sa", """);
    } catch (ClassNotFoundException e) {
        //FIXME unable to find database classes. dosomething
        e.printStackTrace();
    }
catch (SQLException e) {
    //FIXME unable to connect to database dosomething
    e.printStackTrace();
}
return c;
}
C.3.5  WorkerTraceFinal

```java
package kripos.geo;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import jade.content.lang.Codec.CodecException;
import jade.content.lang.sl.SLCodec;
import jade.content.onto.OntologyException;
import jade.content.onto.basic.Action;
import jade.core.behaviours.OneShotBehaviour;
import jade.lang.acl.ACLMessage;
import kripos.ontology.DNSname;
import kripos.ontology.GeoLocation;
import kripos.ontology.IP;
import kripos.ontology.InternetInvestigationsOntology;
import kripos.ontology.TraceResult;
import kripos.ontology.TraceResultList;

public class WorkerTraceFinal extends OneShotBehaviour {
    private static final long serialVersionUID = -605469352616818627L;
    private WorkerAgent myWorkerAgent;
    private Landmark toTrace;
    private double[] myBestLine;
    private ArrayList<Landmark> myLandmarks;

    public WorkerTraceFinal(WorkerAgent a, Landmark toTrace, double[] bestLine, ArrayList<Landmark> landmarks){
        super(a);
        myWorkerAgent = a;
        this.toTrace = toTrace;
        myBestLine = bestLine;
        myLandmarks = landmarks;
    }

    @Override
    public void action() {
        ACLMessage returnResult = new ACLMessage(ACLMessage.INFORM);
        returnResult.addReceiver(myWorkerAgent.getSessionAgent());
        returnResult.setLanguage(new SLCodec(0).getName());
        returnResult.setOntology(InternetInvestigationsOntology.getInstance().getName());
        String convID = myWorkerAgent.getConvList().registerConversation();
        returnResult.setConversationId(convID);
        Action act =
    }
IP target = new IP();
target.setHasName(toTrace.getName());
GeoLocation gl = new GeoLocation();

try{
    Connection con = connectDB();
    Statement st = con.createStatement();
    String getGeoLocation = "SELECT NAME, LATITUDE, LONGITUDE FROM MISC";
    ResultSet rs = st.executeQuery(getGeoLocation);
    while(rs.next()){
        Double lat = rs.getDouble("LATITUDE");
        Double lon = rs.getDouble("LONGITUDE");
        String name = rs.getString("NAME");
        gl.setLocationName(name);
        gl.setLocationLatitude(lat.floatValue());
        gl.setLocationLongitude(lon.floatValue());
    }
}

} catch(SQLException se){
    se.printStackTrace();
}

if(myWorkerAgent.getType().equalsIgnoreCase("TRACE-CBG")){
    TraceResultList trl = new TraceResultList();
    trl.setHasGeoLocation(gl);
    TraceResult tr = new TraceResult();
    tr.setAddressToBeTraced(target); // TODO only first IP address if DNSName
    tr.setHasCaseID(myWorkerAgent.getCaseID());
    tr.setHasSessionID(myWorkerAgent.getSessionID());
    tr.setHasGeoLocation(gl);
    
    Double geoDistance = (toTrace.getMinRTT() - myBestLine[1])/myBestLine[0];
    tr.setTraceResultData(geoDistance.floatValue());
    trl.addTraceResult(tr);
    act = new Action(myWorkerAgent.getSessionAgent(), trl);
}
else if(myWorkerAgent.getType().equalsIgnoreCase("TRACE-GeoPing")){
    TraceResultList trl = new TraceResultList();
    trl.setHasGeoLocation(gl);
    TraceResult targetTr = new TraceResult();
    targetTr.setTraceResultData((new Double(toTrace.getMinRTT()).floatValue()));
    targetTr.setAddressToBeTraced(target);
    trl.addTraceResult(targetTr);
    for (int i=0;i<myLandmarks.size();i++){
        TraceResult tr = new TraceResult();

```java
DNSName name = new DNSName();
name.setName(myLandmarks.get(i).getName());
tr.setAddressToBeTraced(name);
tr.setTraceResultData((new Double(myLandmarks.get(i).
    getMinRTT())).floatValue());
trl.addTraceResult(tr);
}
else {
    TraceResultList trl = new TraceResultList();
    trl.setDoneStatus("failed");
    act = new Action(myWorkerAgent.getSessionAgent(), trl);
}
try {
    myAgent.getContentManager().fillContent(returnResult, act );
    myWorkerAgent.send(returnResult);
    myAgent.doDelete();
} catch (CodecException e) {
    System.out.println("Sending ACL message failed");
    e.printStackTrace();
} catch (OntologyException e) {
    System.out.println("Sending ACL message failed");
    e.printStackTrace();
}
return c;
```
147
148  } //class
C.3.6 PingBehave

```java
package kripos.geo;

import jade.core.behaviours.OneShotBehaviour;
import kripos.geo.ping.HostUnreachableException;
import kripos.geo.ping.MinRTT;

/**
 * Performs actual pinging of remote hosts.
 * Used by WorkerAgents and SessionAgents.
 * @author oysteine
 * @version 1.0
 */
public class PingBehave extends OneShotBehaviour {
    protected static final long serialVersionUID = 9200100880741959482L;
    protected Landmark myTarget;
    protected AgentTemplate myAgentTemplate;
    protected MinRTT myMinRTT;
    protected int myNumPings = 5;
    protected int myPingDistance = 3;
    protected boolean estimateEps = false;
    protected boolean estimateC1 = false;

    /**
     * Uses default values for probeCount and interprobe delay
     * No estimation of epsilon or C1, no rePing()
     * @param a the AgentTemplate this instance belongs to
     * @param landmark the target to ping
     */
    public PingBehave(AgentTemplate a, Landmark target) {
        super(a);
        myAgentTemplate = a;
        myTarget = target;
    }

    /**
     * Uses supplied values
     * TODO currently only IPv4
     * @param a the AgentTemplate this instance belongs to
     * @param landmark the target to ping
     * @param numPings the number of pings to send
     * @param pingDistance the time between pings are sent
     * @param estimateEps true if epsilon is to be estimated
     * @param estimateC1 true if C1 is to be estimated
     */
    public PingBehave(AgentTemplate a, Landmark target, int numPings,
```
```java
int pingDistance, boolean estimateEps, boolean estimateC1 {
    super(a);
    myAgentTemplate = a;
    myTarget = target;
    myNumPings = numPings;
    myPingDistance = pingDistance;
    this.estimateEps = estimateEps;
    this.estimateC1 = estimateC1;
}

/*@non-Javadoc*/
* @see jade.core.behaviours.Behaviour#action()
*/
@Override
public void action() {
    myMinRTT = new MinRTT();
    try {
        myMinRTT.ping(myTarget.getIP(), myNumPings,
            myPingDistance, false);
        if(estimateC1){
            myMinRTT.computeCRegions(true);
            myTarget.setC1(myMinRTT.getCI());
        }
        if(estimateEps){
            myTarget.setEpsilon(myMinRTT.estimateEpsilon());
        }
        myTarget.setMinRTT(myMinRTT.getMinRTT());
    }
    catch (HostUnreachableException e) {
    }
}
```
C.4 GWAgent Classes

C.4.1 GWAgent

```java
package kripos.gateway;

import java.util.ArrayList;
import java.util.GregorianCalendar;
import java.util.Random;
import java.util.concurrent.ConcurrentHashMap;
import jade.content.ContentElement;
import jade.content.lang.Codec.CodecException;
import jade.content.lang.sl.SLCodec;
import jade.content.onto.OntologyException;
import jade.content.onto.UngroundedException;
import jade.content.onto.basic.Action;
import jade.content.onto.basic.Result;
import jade.core.AID;
import jade.core.ContainerID;
import jade.domain.DFService;
import jade.domain.FIPAException;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.FIPAManagementOntology;
import jade.domain.FIPAAgentManagement.SearchConstraints;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.domain.JADEAgentManagement.WhereIsAgentAction;
import jade.domain.mobility.MobilityOntology;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.ConversationList;
import jade.lang.acl.MessageTemplate;
import jade.wrapper.gateway.GatewayAgent;
import kripos.ontology.InternetInvestigationsOntology;

/**
 * @author oysteine
 * @version 1.0
 */
public class GWAgent extends GatewayAgent {
    private static final long serialVersionUID = -5814890860284035720L;
    protected ConversationList convList;
    protected GregorianCalendar myCal;
    protected ConcurrentHashMap<AID, String> adminAgents = new ConcurrentHashMap<AID, String>();
    private ConcurrentHashMap<String, CommandPackage> activeCommands = new ConcurrentHashMap<String, CommandPackage>();

    /**
     * @param
     */
    public GWAgent() {
```
        convList = new ConversationList(this);
        myCal = new GregorianCalendar(); // timezone, not implemented
    }

    @Override
    protected void setup()
    {
        getContentManager().registerLanguage(new SLCodec(0));
        getContentManager().registerLanguage(new SLCodec()); // some messages from AMS not set to SL-0
        getContentManager().registerOntology(MobilityOntology.getInstance());
        getContentManager().registerOntology(InternetInvestigationsOntology.getInstance());
        getContentManager().registerOntology(FIPAManagementOntology.getInstance());

        // find AdminAgents
        DFAgentDescription adminTemplate = new DFAgentDescription();
        ServiceDescription adminSd = new ServiceDescription();
        adminSd.setType("AdminAgent");
        adminTemplate.addServices(adminSd);
        SearchConstraints sc = new SearchConstraints();
        sc.setMaxDepth(new Long(0));
        sc.setMaxResults(new Long(10000));
        ArrayList<DFAgentDescription> tempList = new ArrayList<DFAgentDescription>();

        try {
            DFAgentDescription dfas[] = DFService.search(this, adminTemplate, sc);
            for(int i=0; i<dfas.length;i++){
                tempList.add(dfas[i]);
            }
        }
        catch (FIPAException e1) {
            e1.printStackTrace();
        }

        for(int i=0;i< tempList.size();i++){
            WhereIsAgentAction wiaa = new WhereIsAgentAction();
            wiaa.setAgentIdentifier(tempList.get(i).getName());
            AID myAMS = getAMS();
            Action act = new Action(myAMS, wiaa);
            ACLMessage qMsg = new ACLMessage(ACLMessage.REQUEST);
            String convID = getName() + new Random().nextLong();
            qMsg.setConversationId(convID);
            qMsg.setLanguage(new SLCodec(0).getName());
            qMsg.setOntology(MobilityOntology.getInstance().getName());
        }
try {
    getContentManager().fillContent(qMsg, act);
    qMsg.addReceiver(act.getActor());
    // register conversation with agent to get correct reception
    getConvList().registerConversation(convID);
    send(qMsg);
    MessageTemplate mt = MessageTemplate.
        MatchConversationId(convID);
    ACLMessage resp = blockingReceive(mt, 1000000);
    getConvList().deregisterConversation(convID);
    ContentElement ce = getContentManager().extractContent( resp);
    Result result = (Result) ce;
    ContainerID cid = (ContainerID)result.getValue();
    adminAgents.put(tempList.get(i).getName(), cid.getName( ));
}
catch (UngroundedException e) {
    e.printStackTrace();
} catch (CodecException e) {
    e.printStackTrace();
} catch (OntologyException e) {
    e.printStackTrace();
}
}
AID df = getDefaultDF();
ACLMessage adminSubs = DFServe.createSubscriptionMessage( this, df, adminTemplate, sc);
convList.registerConversation(adminSubs.getConversationId() );
addBehaviour(new AdminSubscribeBehaviour(this, adminSubs, adminAgents));
addBehaviour(new GWReceiveBehaviour(this));
/**
 * *
 */
@Override
protected void processCommand(java.lang.Object command){
    if (command instanceof CommandPackage){
        CommandPackage cp = (CommandPackage)command;
        addBehaviour(new LaunchTraceBehave(this, cp));
    } else {
        // todo throw exception to alert gw and webapp/user of error
        System.out.println("erereorerer");
        releaseCommand(command);
    
}
/**
 * Get the CommandPackage associated with the conversation ID supplied
 * Remove the CommandPackage from the map
 * Deregisters the conversation ID with the agent
 * @param conversationID to match
 * @return commandpackage associated with conversationID
 */
public CommandPackage getMatchingCommand(String conversationID){
    convList.deregisterConversation(conversationID);
    return activeCommands.remove(conversationID);
}
/**
 * Adds an entry in the map of activeCommands
 * @param convID
 * @param cp
 */
public void addCommand(String convID, CommandPackage cp){
    activeCommands.put(convID, cp);
}
/**
 * Returns the conversationlist of this agent
 * @return the ConversationList of this agent
 */
protected ConversationList getConvList(){
    return convList;
}
/**
 * Returns the GregorianCalendar of this agent
 * @return the calendar of this agent
 */
protected GregorianCalendar getCalendar(){
    return myCal;
}
C.4.2 GWReceiveBehaviour

```java
class GWReceiveBehaviour extends CyclicBehaviour {
    private static final long serialVersionUID = 201428984738025171L;
    private GWAgent myGWAgent;

    public GWReceiveBehaviour(GWAgent a) {
        super(a);
        myGWAgent = a;
    }

    @Override
    public void action() {
        MessageTemplate mt = myGWAgent.getConvList().getMessageTemplate();
        ACLMessage msg = myAgent.receive(mt);
        if (msg != null) {
            handle(msg);
        }
        block(100); // this SHOULD work without timeout
        // block();
    }
}
```
* If the received action is not understood a NOT_UNDERSTOOD message is returned to the sender.

* @param msg incoming {@link jade.lang.acl.ACLMessage}

*/
private void handle(ACLMessage msg){
    try {
        ContentElement content = myAgent.getContentManager().extractContent(msg);
        Concept action = ((Action)content).getAction();
        if (action instanceof TraceResultList){
            TraceResultList trl = (TraceResultList)action;
            CommandPackage cp = myGWAgent.getMatchingCommand(msg.getConversationId());
            ArrayList< TraceResult > resList = new ArrayList< TraceResult >();
            for(int i=0;i<trl.getTraceResult().size();i++){
                resList.add( (TraceResult)trl.getTraceResult().get(i) );
            }
            if(cp.getType().equalsIgnoreCase("TRACE-CBG")){
                cp.setCBGResults(resList);
            } else if (cp.getType().equalsIgnoreCase("TRACE-GEOPING")){
                cp.setGeoPingResults(resList);
            }
            cp.setRepTime(myGWAgent.getCalendar().getTime());
            if(trl.getDoneStatus().equalsIgnoreCase("OK")){
                cp.setSuccessful(true);
            } else{
                cp.setSuccessful(false);
            }
            myGWAgent.releaseCommand(cp);
        } else if (action instanceof UpdateTime){
            UpdateTime upd = (UpdateTime)((Action)content).getAction();
            long offset = Math.round(upd.getHasOffset());
            myGWAgent.getCalendar().setTimeInMillis(offset + myGWAgent.getCalendar().getTimeInMillis());
            //log to SysLog
        } else {
            ACLMessage reply = msg.createReply();
            msg.setPerformative(ACLMessage.NOT_UNDERSTOOD);
            myAgent.send(reply);
        }
    } catch (UngroundedException e) {
        e.printStackTrace();
    }
}
catch (CodecException e) {
    e.printStackTrace();
}

catch (OntologyException e) {
    e.printStackTrace();
}

} // class
C.4.3 LaunchTraceBehave

```java
package kripos.gateway;

import java.math.BigInteger;
import java.util.GregorianCalendar;
import java.util.Iterator;
import java.util.Set;
import jade.content.lang.Codec.CodecException;
import jade.content.lang.sl.SLCodec;
import jade.content.onto.OntologyException;
import jade.content.onto.basic.Action;
import jade.core.AID;
import jade.core.behaviours.SimpleBehaviour;
import jade.domain.mobility.MobilityOntology;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import kripos.ontology.*;

/**
 * Launched by GWAgent when a CommandPackage containing instructions to commence a trace is received
 * @author oysteine
 * @version 1.0
 */
public class LaunchTraceBehave extends SimpleBehaviour {
    private static final long serialVersionUID = 4144187902498096653L;
    private CommandPackage command;
    private GWAgent myGWAgent;
    private boolean done;

    /**
     * Creates an instance of this behaviour
     * @param a the GWAgent owning this instance
     * @param the CommandPackage containing information about what to do
     */
    public LaunchTraceBehave(GWAgent a, CommandPackage cp) {
        super(a);
        command = cp;
        myGWAgent = a;
    }

    /**
     * (non-Javadoc)
     * @see jade.core.behavioursBehaviour#action()
     */
    public void action() {
        // parse package
        command.getReqTime();
        command.getTarget();
    }
}
```
50 command.getType();
51
// construct and fill CreateCase, Case and Session
52 CreateCase cc = new CreateCase();
53 OntAddress target;
54 if (isIP()) {
55   IP ip = new IP();
56   ip.setHasName(command.getTarget());
57   target = ip;
58 }
59 else {
60   DNSname dns = new DNSname();
61   dns.setHasName(command.getTarget());
62   target = dns;
63 }
64 Case c = new Case();
65 c.setCaseID((int) Math.round(Math.random() * 100000000));
66 c.setCaseName("testcase"); // todo get from GUI
67
68 Session s = new Session();
69 s.setCaseID(c.getCaseID());
70 s.setHasBaseAddress(target);
71 c.setCaseSessions(s);
72 cc.setHasCase(c);
73
74 OntDate date = new OntDate();
75 long time = GregorianCalendar.getInstance().getTimeInMillis();
76 date.setTime(time);
77 cc.setHasDate(date);
78 c.setCaseStartDate(date);
79
80 Action act = new Action(getRandomAdminAgent(), cc);
81 String convID = sendRequest(act);
82 myGWAgent.addCommand(convID, command);
83 MessageTemplate mt = MessageTemplate.MatchConversationId(convID);
84 myAgent.blockingReceive(mt);
85 }
86
87 /**
88 * @return true if the String represents and IP address and not DNS name
89 */
90 private boolean isIP()
91 {
92   String toParse = command.getTarget().replace('.', '0');
93   try{
94     new BigInteger(toParse);
95     return true;
96   }
97   catch (NumberFormatException nfe){
return false;
}

/**
 * Get a random AdminAgent from the hashmap of AdminAgents
 *
 * @return the AID of the selected AdminAgent
 */
private AID getRandomAdminAgent(){
    int size = myGWAgent.adminAgents.size();
    int random = (int)Math.round((Math.random())*size);
    Set<AID> tempSet = myGWAgent.adminAgents.keySet();
    Iterator<AID> it = tempSet.iterator();
    for(int i=0;i<random-1;i++){
        it.next();
    }
    return it.next();
}

/**
 * Utility method for sending ACLMessages
 * <p>
 * Registers the conversationID in the convList
 * to avoid loosing any reply
 *<p>
 * @param action the action that is to be wrapped in an
 * ACLMessage
 * @return the conversationID for the created ACLMessage
 */
private String sendRequest(Action action){
    try {
        ACLMessage qMsg = new ACLMessage(ACLMessage.REQUEST);
        String convID = myGWAgent.getConvList().registerConversation();
        qMsg.setConversationId(convID);
        qMsg.setLanguage(new SLCodec(0).getName());
        qMsg.setOntology(MobilityOntology.getInstance().getName());
        myAgent.getContentManager().fillContent(qMsg, action);
        qMsg.addReceiver(action.getActor());
        //register conversation with agent to get correct
        //reception of reply
        myAgent.send(qMsg);
        return convID;
    }
    catch (CodecException e) {
        e.printStackTrace();
        return null;
    }
    catch (OntologyException e) {
        e.printStackTrace();
        return null;
    }
}
@see jade.core.behaviours.Behaviour#done()
package kripos.gateway;

import java.util.concurrent.ConcurrentHashMap;
import jade.content.ContentElement;
import jade.content.lang.Codec.CodecException;
import jade.content.lang.sl.SLCodec;
import jade.content.onto.OntologyException;
import jade.content.onto.UngroundedException;
import jade.content.onto.basic.Action;
import jade.content.onto.basic.Result;
import jade.core.AID;
import jade.core.ContainerID;
import jade.domain.DFService;
import jade.domain.FIPAException;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.mobility.MobilityOntology;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import jade.proto.SubscriptionInitiator;

/**
 * Subscribes to changes in registered AdminAgents by the AMS Service.
 * Any changes are recorded in a ConcurrentHashMap that was first initialized by setup().
 * @author oysteine
 * @version 1.0
 */
public class AdminSubscribeBehaviour extends SubscriptionInitiator {

private static final long serialVersionUID = 988748612411732543L;
private GWAgent myAgent;
private ConcurrentHashMap<AID, String> myMap;

/**
 * Constructor
 * @param a The agent this instance belongs to
 * @param msg The message used by the behaviour to initiate subscription
 * @param map The hashmap for recording changes.
 */
public AdminSubscribeBehaviour(GWAgent a, ACLMessage msg,
ConcurrentHashMap<AID, String> map) {
    super(a, msg);
    myAgent = a;
    myMap = map;
}
/**
 * Handles incoming inform messages that contains changes in
 * registered AdminAgents.
 * The AMS is queried for the Location of any new AdminAgents.
 * Removed AdminAgents are delete from the hashmap.
 */

@Override
protected void handleInform(ACLMessage inform) {
  try {
    DFAgentDescription[] dfds =
      DFService.decodeNotification(inform.getContent());
    for (int i = 0; i < dfds.length; i++) {
      AID aid = dfds[i].getName();

      if (myMap.containsKey(aid)) {
        /* There seems to be a problem with the AMS, messages
         * containing removal information are sent without reason. This
         * functionality disabled.
         */
        myMap.remove(aid);
      } else {
        WhereIsAgentAction wiaa = new WhereIsAgentAction();
        wiaa.setAgentIdentifier(aid);
        AID myAMS = myAgent.getAMS();
        Action act = new Action(myAMS, wiaa);

        ACLMessage qMsg = new ACLMessage(ACLMessage.REQUEST);
        String convID = myAgent.getConvList().registerConversation();
        qMsg.setConversationId(convID);
        qMsg.setLanguage(new SLCodec(0).getName());
        qMsg.setOntology(MobilityOntology.getInstance().getName());

        try {
          myAgent.getContentManager().fillContent(qMsg, act);
          qMsg.addReceiver(act.getActor());
          // register conversation with agent to get correct reception
          myAgent.send(qMsg);
        }

        MessageTemplate mt = MessageTemplate.MatchConversationId(convID);
        ACLMessage resp = myAgent.blockingReceive(mt, 1000000);
        myAgent.getConvList().deregisterConversation(convID);
        ContentElement ce = myAgent.getContentManager().extractContent(resp);
        Result result = (Result) ce;
      }
    }
  }
}
ContainerID cid = (ContainerID)result.getValue();
myMap.put(aid, cid.getName());

} catch (UngroundedException e) {
e.printStackTrace();
}
} catch (CodecException e) {
e.printStackTrace();
}
} catch (OntologyException e) {
e.printStackTrace();
}
} catch (FIPAException fe) {
fe.printStackTrace();

} //class
C.4.5 CommandPackage

// part of design
package kripos.gateway;
import java.util.Date;
import java.util.ArrayList;
import kripos.geo.DelayVector;
import kripos.geo.Landmark;
import kripos.ontology.TraceResult;
/**
 * Class for wrapping all necessary information in a bundle for easy exchange between the multi-agent system and the gateway-classes.
 * @author oysteine
 * @version 1.0
 */
public class CommandPackage {
  private String target; // the target of the trace
  private String type; // trace type. may be extended to also cover securing of content etc
  private Date reqTime; // when was the trace requested
  private Date repTime; // when was the trace finalized
  private boolean successful = false; // the final state of the trace
  private ArrayList<Landmark> geoPingResults;
  private ArrayList<TraceResult> CBGResults = new ArrayList<TraceResult>();
  private ArrayList<DelayVector> delayVectors = new ArrayList<DelayVector>();

  /**
   * Construct an empty CommandPackage.
   * Any fields must be filled by the relevant set methods.
   */
  public CommandPackage() {
  }

  public void addResult(TraceResult tr){
    CBGResults.add(tr);
  }

  /**
   * @return the CBGResults
   */
  public ArrayList<TraceResult> getCBGResults() {
    return CBGResults;
  }
}
```java
/**
 * @param results the cBGResults to set
 */
public void setCBGResults(ArrayList<TraceResult> results) {
    CBGResults = results;
}

/**
 * @return the geoPingResults
 */
public ArrayList getGeoPingResults() {
    return geoPingResults;
}

/**
 * @param geoPingResults the geoPingResults to set
 */
public void setGeoPingResults(ArrayList geoPingResults) {
    this.geoPingResults = geoPingResults;
}

/**
 * @return the repTime
 */
public Date getRepTime() {
    return repTime;
}

/**
 * @param repTime the repTime to set
 */
public void setRepTime(Date repTime) {
    this.repTime = repTime;
}

/**
 * @return the reqTime
 */
public Date getReqTime() {
    return reqTime;
}

/**
 * @param reqTime the reqTime to set
 */
public void setReqTime(Date reqTime) {
    this.reqTime = reqTime;
}
```
/**
 * @return the successful
 */
public boolean isSuccessful() {
    return successful;
}

/**
 * @param successful the successful to set
 */
public void setSuccessful(boolean successful) {
    this.successful = successful;
}

/**
 * @return the target
 */
public String getTarget() {
    return target;
}

/**
 * @param target the target to set
 */
public void setTarget(String target) {
    this.target = target;
}

/**
 * @return the type
 */
public String getType() {
    return type;
}

/**
 * @param type the type to set
 */
public void setType(String type) {
    this.type = type;
}

public ArrayList<DelayVector> getDelayVectors() {
    return delayVectors;
}
156 } // class
C.5 Ping Classes

C.5.1 MinRTT

```java
package kripos.geo.ping;

// TODO support ping4 and ping6 simultaneously

/*
 * Some of the functionality of this class is adapted from
 * RTTometer.
 * (Code originally released under GPLv2)
 * The RTTometer application is available from: http://idmaps.eecs.umich.edu/rttometer/
 * The GPLv2 is available at: http://www.gnu.org/copyleft/gpl.html
 ***/

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.nio.charset.Charset;
import java.util.ArrayList;
import java.util.Arrays;

/**
 * @author oysteine
 * @version 1.0
 */
public class MinRTT {
    private float CI = 0;
    private float CII = 0;
    private float CIII = 0;
    private ArrayList<PingItem> probeList = new ArrayList<PingItem>();
    private float minimumRTT = 99999.0f;
    private float maximumRTT = 0;
    private float epsilon = MINEPS; // automatically changed based on minimumRTT

    /* default values of epsilon */
    public final static float UNINETTEPS = 0.7f; // single AS, most paths <<20ms
    public final static float MINEPS = 2; // <=50ms
    public final static float MEDEPS = 4; // 50-150ms
    public final static float BIGEPS = 6; // >150ms
    private float CI_threshold = 0.8f; // confidence in CI
    private final int EPS_MIN_PROBES = 100; // magic number. only for measuring between landmarks
    private boolean estimate_CI_confidence;
```
private int listCounter = 0; //used to resume calculations if the probelst is extended
private float accCi = 0; //cumulative
private float accCii = 0; //cumulative
private float accCiii = 0; //cumulative
private int accN = 0; //cumulative
private int numProbes; //Updated if successive runtimes
private int rePingCnt = 0;
// used by rePing()
private String previousAddress;
private int previousNumPings;
private double previousPingDistance;

/**
 * @param address
 * @param numPings
 * @param pingDistance
 * @throws HostUnreachableException
 */
public MinRTT(){
}

/**
 * @param address
 * @param numPings
 * @param pingDistance
 * @throws HostUnreachableException
 */
public ArrayList ping(String address, int numPings, double pingDistance,
boolean reset) throws HostUnreachableException{
    if(reset){
        reset();
    }
    previousAddress = address;
    previousNumPings = numPings;
    previousPingDistance = pingDistance;
    int lastSequence = 0;
    listCounter = 0;
    try {
        Runtime runtime = Runtime.getRuntime();
        String command = "ping\-i"+pingDistance+"\-c"+numPings+"\n" + address;
        Process p = runtime.exec(command);
        numProbes += numPings;
        InputStream ip = p.getInputStream();
        InputStreamReader isr = new InputStreamReader(ip, Charset .forName("ISO-8859-1"));
        BufferedReader br = new BufferedReader(isr);
        String temp = null;
        while((temp = br.readLine()) !=null){
// handle unreachable host
if(temp.contains("0\nreceived, 100% packet loss")){
  for(int i=0; i<numPings; i++){
    probeList.add(new PingItem(PingItem.LOST, lastSequence+i+1, -999999));
    listCounter++;
  }
  throw new HostUnreachableException("100% packet loss");
}

// sort out filtered replies
if(temp.contains("Packet filtered")){
  int cutSub = temp.indexOf("icmp_seq=");
  String subTemp = temp.substring(cutSub);
  int cutSeq = subTemp.indexOf('=');
  int cutSeqEnd = subTemp.indexOf("Packet filtered");
  String seqString = subTemp.substring(cutSeq+1, cutSeqEnd-1);
  int seq = Integer.parseInt(seqString);

  int seqDelta = seq - lastSequence;
  if(seqDelta > 1){
    for(int i=0; i<seqDelta; i++){
      PingItem lostPi = new PingItem(PingItem.LOST, lastSequence+i+1, -999999);
      probeList.add(lostPi);
      listCounter++;
    }
  }

  lastSequence = seq;
  PingItem pi = new PingItem(PingItem.FILTERED, seq, -999999);
  probeList.add(pi);
  listCounter++;
  throw new HostUnreachableException("Packet filtering on path to host");
}

// normal ping item, if not 64 bytes it is not a regular // ICMP_ECHO_REPLY packet and we ignore it
if(temp.contains("64 bytes from")){
  int cutPoint = temp.indexOf("=");
  String cutTemp = temp.substring(cutPoint);
  int seqEndPoint = cutTemp.indexOf("ttl");
  String seqString = cutTemp.substring(1, seqEndPoint-1);
  int seq = Integer.parseInt(seqString);

  int seqDelta = seq - lastSequence;
  if(seqDelta > 1){
    for(int i=0; i<seqDelta; i++){
      PingItem lostPi = new PingItem(PingItem.LOST, lastSequence+i+1, -999999);
      probeList.add(lostPi);
      listCounter++;
    }
  }

  lastSequence = seq;
  PingItem pi = new PingItem(PingItem.FILTERED, seq, -999999);
  probeList.add(pi);
  listCounter++;
  throw new HostUnreachableException("Packet filtering on path to host");
}
for(int i=0; i<seqDelta;i++){
    PingItem lostPi = new PingItem(PingItem.LOST,
        lastSequence+i,-999999);
    probeList.add(lostPi);
    listCounter++;
}
lastSequence = seq;
int timePoint = temp.lastIndexOf("=");
int endPoint = temp.lastIndexOf("ms");
String time = temp.substring(timePoint+1, endPoint-1) ;
        float rtt = Float.parseFloat(time);
        PingItem pi = new PingItem(PingItem.NORMAL_RTT, seq,
            rtt);
        probeList.add(pi);
        listCounter++;
}
if(temp.contains("rtt\_min/avg/max/mdev\_\=")){
    //common variables
    int cutPoint = temp.indexOf("=");
    String cutTemp = temp.substring(cutPoint+2);
    int firstSlashPoint = cutTemp.indexOf("/");
    //maxRTT //not needed
    int maxTempEndPointLong = cutTemp.indexOf("ms");
    String maxTempLong = cutTemp.substring(0,
        maxTempEndPointLong);
    int maxTempEndPoint = maxTempLong.lastIndexOf(" /");
    String maxTemp = maxTempLong.substring(0,
        maxTempEndPoint);
    int maxEndPoint = maxTemp.lastIndexOf("/");
    String max = maxTemp.substring(maxEndPoint+1,
        maxEndPoint);
    float maxRtt = Float.parseFloat(max);
    if (maximumRTT < maxRtt){
        maximumRTT = maxRtt;
    }
    //minRTT
    String min = cutTemp.substring(0, firstSlashPoint);
    float minRtt = Float.parseFloat(min);
    if(minimumRTT > minRtt){
        minimumRTT = minRtt;
    }
} //if
} //while
} //try
catch (IOException e) {
    throw new HostUnreachableException(e.getMessage(), e.getCause());
}
return probeList;

/**
 * Method adapted from RTTOmeter application
 */
public int computeCRegions(boolean estimateConfidence) {
    estimate_CI_confidence = estimateConfidence;
    float ci = 0;
    float cii = 0;
    float ciIII = 0;
    int n = 0; //total number of phaseplot points (not probes)
    float val = 0;
    float w1 = 0;
    float w2 = 0;
    float w = 0;
    int loopStart = 0;
    if (minimumRTT < 20) {
        epsilon = UNINETTEPS;
    } else if (minimumRTT < 50) {
        epsilon = MINEPS;
    } else if (minimumRTT < 150) {
        epsilon = MEDEPS;
    } else {
        epsilon = BIGEPS;
    }
    if (probeList.size() != listCounter) {
        loopStart = (probeList.size() - listCounter) - 1; //include the last probe from the previous run
    }
    for (int i = loopStart; i < probeList.size() - 1; i++) {
        PingItem piCurrent = probeList.get(i);
        PingItem piNext = probeList.get(i + 1);

        //Classify the point in the phase-plot
        if (piCurrent.getType() == PingItem.NORMAL_RTT && piNext.getType() == PingItem.NORMAL_RTT) {
            n++; // we do not want to compare the last pingitem to null
        }
        if (piCurrent.getType() == PingItem.NORMAL_RTT && piNext.getType() == PingItem.NORMAL_RTT) {
            n++;
        } /* Point in CI, CII, or CIII */
        if ((piCurrent.getRTT() - minimumRTT) <= epsilon) {
w1 = 1.0f;
else {
    val = ((int)((piCurrent.getRTT() - minimumRTT)/
        epsilon) + 1);
    w1 = 1.0f/val;
}

if( (piNext.getRTT() - minimumRTT) <= epsilon){
    w2 = 1.0f;
} else {
    val = ((int)((piNext.getRTT() - minimumRTT)/epsilon) + 1);
    w2 = 1.0f/val;
}

w = w1 * w2;
}
else {
    /* Point in CII or CIII */
    if(piCurrent.getType() == PingItem.LOST) { 
        w1 = 0.0f;
    } else {
        val = ((int)((piCurrent.getRTT() - minimumRTT)/
            epsilon) + 1);
        w1 = 1.0f/val;
    }
    
    if(piNext.getType() == PingItem.LOST) {
        w2 = 0.0f;
    } else {
        val = ((int)((piNext.getRTT() - minimumRTT)/epsilon) + 1);
        w2 = 1.0f/val;
    }

    w = w1 * w2;
}

if(w1 == 1 || w2 == 1){
    /* This point in CII */
    cii += (1-w);
} else{
    /* Definitely CIII */
    ciii += (1-w);
}

accCi+=ci;
accCii+=cii;
accCiii+=ciii;
accN+=n;
if(accN > 0) {
    CI = accCi/accN;
    CII = accCii/accN;
    CIII = accCiii/accN;
}
if(estimate_CI_confidence && CI < CI_threshold && rePingCnt <= 2){
    rePing();
} return n;
}//compute

private void rePing() {//reuses original input. TODO: estimate better values. wait() if c2 high?
    rePingCnt++;
    try {
      ping(previousAddress, previousNumPings, previousPingDistance, false);
      computeCRegions(estimate_CI_confidence);
    } catch (HostUnreachableException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
}

private void reset(){
    accCi = 0;
    accCii = 0;
    accCiii = 0;
    accN = 0;
    rePingCnt = 0;
    numProbes = 0;
    CI = 0;
    CII = 0;
    CIII = 0;
    probeList.clear();
    minimumRTT = 99999.0f;
    maximumRTT = 0;
}

/**
   * Estimates epsilon from the ping measurements
   * Method adapted from RTTometer application
   */
public float estimateEpsilon(){
    int n = 0;
    float eps = -1.0f;
float[] rtts = new float[numProbes];

/* make array of rtts from probelist */
for (int i=0;i<probeList.size();i++) {
    PingItem piCurrent = probeList.get(i);
    if (piCurrent.getType() == PingItem.NORMAL_RTT) {
        rtts[n++] = piCurrent.getRTT();
    }
}

if (n >= (0.8 * EPS_MIN_PROBES)) { /* At least 80% of the probes are successful */
    Arrays.sort(rtts);
    ModeNode mn = new ModeNode();
    float mode_all = mn.mode(rtts, n);
    /* Estimate epsilon */
    eps = 2*(mode_all - rtts[0]); // or use minimumRTT, same value
    /* Make sure eps >=0, mode may be very close to min, */
    /* and due to precision the subtraction might give */
    /* negative result */
    if (eps < 0) {
        eps = 0.0f;
    }
    return eps;
}

/*
Getter methods
*/
public float getCI() {
    return CI;
}

public float getCII() {
    return CII;
}

public float getCIII() {
    return CIII;
}

public float getMaxRTT() {
    return maximumRTT;
}

public float getMinRTT() {
    return minimumRTT;
}
} // class
C.5.2 ModeNode

```java
package kripos.geo.ping;

/*
 * Most of the functionality of this class is adapted from RTTOmeter.
 * While the main RTTOmeter application is released under the GPLv2,
 * the file mode.c is apparently not (breach of GPLv?).
 * The notice below is required by the original author.
 */

import java.util.Hashtable;

public class ModeNode {
    private Long key;
    private long cnt;

    /**
     * @param x
     * @return
     */
    private int ceiling(float x){
        if((int)(Math.abs(x) + 0.5) > (int)(Math.abs(x)) ){
            return (int)(Math.abs(x+0.5));
        }
    }
}
```

* Copyright (c) 2003, Amgad Zeitoun.
* All rights reserved.
* * Redistribution and use in source and binary forms are permitted
* provided that the above copyright notice and this paragraph are
* duplicated in all such forms and that any documentation,
* advertising materials, and other materials related to such
* distribution and use acknowledge that the software was developed
* by Amgad Zeitoun at the University of Michigan, Ann Arbor.
The * name of the University may not be used to endorse or promote
* products derived from this software without specific prior
* written permission.
* THIS SOFTWARE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR
* IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
* WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
* PURPOSE.
* * Author: Amgad Zeitoun (azeitoun@eecs.umich.edu)
*/

```
```java
44     }
45     else {
46         return (int)(Math.abs(x));
47     }
48     }
49
50     /**
51     * @param m1
52     * @param m2
53     * @return
54     */
55     public int matchNode(ModeNode m1, ModeNode m2){
56         if(m1.getKey() < m2.getKey()) {
57             return -1;
58         }
59         if(m1.getKey() > m2.getKey()) {
60             return 1;
61         }
62         return 0;
63     }
64
65
66     /**
67     * Calculate the mode of an array of values with length len
68     * @param array
69     * @param len
70     * @return
71     */
72     public float mode(float[] array, int len){
73         float mode = -1.0f;
74         long max_cnt = 0;
75         long val = 0;
76         boolean dbl_precision;
77         long key;
78         if (array.length <= 0){
79             return mode;
80         }
81         Hashtable<Long,ModeNode> hashtable = new Hashtable<Long, ModeNode>(array.length);
82         /* If the RTTs are very small (i.e., less than 1 ms), make
double digitis mode
precision, otherwise make it a single digit
precision */
83         /* NOTE: I assume that array is sorted. Which is true,
because I call mode() after I qsort the array */
84         if (array[0] < 1.0){
194
```
dbl_precision = true;
}
else{
    dbl_precision = false;
}

for (float element : array) {
    /* Using a single digit precision by default, except when
       the values
       of RTTs are really small (<1.0ms) */
    if(dbl_precision){
        key = ceiling(element * 100);
    }
    else{
        key = ceiling(element * 10);
    }
    ModeNode m = hashtable.get(key);
    if(m == null) {
        m = new ModeNode();
        m.setKey(key);
        hashtable.put(m.getKey(), m);
    }
    m.incCnt();
    if(m.getCnt() > max_cnt) {
        /* we have a mode here */
        max_cnt = m.getCnt();
        val = key;
    }
    /* TODO: detect multiple modes */ //dette fikser vi
}
/* The mode should be repeated more than once! */
/* Just in case we don’t have any mode at all */
if ( max_cnt > 1 ) {
    if(dbl_precision){
        mode = (float)val/100;
    }
    else{
        mode = (float)val/10;
    }
}
return mode;

public void incCnt(){
public long getCnt(){
    return cnt;
}

public Long getKey(){
    return key;
}

public void setKey(Long newKey){
    key = newKey;
}

} //class
package kripos.geo.ping;

public class PingItem {

public final static int FILTERED=-2;
public final static int LOST=-1;
public final static int NORMAL_RTT=0;
public final static int MIN_RTT=1;//not needed
public final static int MAX_RTT=2;//not needed
public final static int AVG_RTT=3;//not needed
public final static int MED_RTT=4;//not needed

private final int type;
private final int sequence;
private final float rtt;

/**
 * @param newType
 * @param seqnr
 * @param rtt
 */
public PingItem(int newType, int seqnr, float rtt){
type = newType;
sequence = seqnr;
this.rtt = rtt;
}

/**
 * Getter methods
 */

/**
 * The type indicates if this PingItem represents
 * real ping information or a lost packet
 *
 * @return the type of this PingItem
 */
public int getType(){
    return type;
}

/**
 * The sequence number only has local meaning.
 *
 * @return the sequence number of this PingItem
 */
public int getSequence(){
    return sequence;
}

/**
 * @return the RTT of this PingItem

/ *
54  public float getRTT(){
55      return rtt;
56  }
57
58 } // class
package kripos.geo.ping;

public class HostUnreachableException extends Exception {

    private static final long serialVersionUID = -4310128500792634318L;

    public HostUnreachableException() {
    }

    /**
     * @param message
     */
    public HostUnreachableException(String message) {
        super(message);
    }

    /**
     * @param cause
     */
    public HostUnreachableException(Throwerable cause) {
        super(cause);
    }

    /**
     * @param message
     * @param cause
     */
    public HostUnreachableException(String message, Throwable cause) {
        super(message, cause);
    }
}

}//class
C.6 Servlet Classes

C.6.1 GetTrace

```java
package kripos.math.servlet;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/**
 * Display a page where an IP to trace can be enter.
 * Display a map with trace information if an IP has been entered.
 *
 * @author oyesteine
 */
public class GetTrace extends HttpServlet {

    private static final long serialVersionUID =
    6720480250134837969L;

    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
        String ip = request.getParameter("ip");

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Trace IP</title>");
        out.println("<body>");

        out.println("<h3>Perform tracing</h3>");
        out.println("<form action="Map" method=POST>");
        out.println("IP:");
        out.println("<input type=text size=20 name=ip>");
        if (ip != null) {
            out.println("value=" + ip + ">");
        }
        out.println("</form>");
        out.println("<input type=submit name=action value="Get trace"> ");
        out.println("<input type=submit name=action value="Force trace"> ");
```

200
        out.println("</form>");
46
47        out.println("</body>");
48        out.println("</html>");
49    }
50
51    @Override
52    public void doPost(HttpServletRequest request,
53             HttpServletResponse response)
54    throws IOException, ServletException
55    {
56        doGet(request, response);
57    }
C.6.2 GetMap

```java
package kripos.math.servlet;

import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.IOException;
import java.io.PrintStream;
import java.io.PrintWriter;
import java.util.Locale;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import kripos.geo.openmap.IntersectionInfo;
import kripos.geo.openmap.MapDrawer;

/**
 * Display a page with the results of a trace.
 * The trace is performed if not cached already.
 *
 * @author oysteine
 */
public class GetMap extends HttpServlet {
    private static final long serialVersionUID = 7631170050282471534L;

    private static String BASE_PATH = null;

    /**
     * @return path for OpenMap data. Does <i>not</i> end with separator.
     *
     */
    public static String getDataPath() {
        String path = BASE_PATH;
        if (path == null) {
            path = System.getProperty("user.dir") + File.separator + "data";
        }
        return path;
    }

    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException {
```
ServletContext sc = getServletContext();
if (BASE_PATH == null) {
    BASE_PATH = sc.getRealPath(""") + File.separator + "data";
}

String ip = request.getParameter("ip");
String cacheId = request.getParameter("cacheId");
String strZoom = request.getParameter("zoom");
String strLongOffset = request.getParameter("longOffset");
String strLatOffset = request.getParameter("latOffset");
float zoom = 1.0f;
if (strZoom != null) {
    zoom = Float.parseFloat(strZoom);
}
double longOffset = 0.0;
if (strLongOffset != null) {
    longOffset = Double.parseDouble(strLongOffset);
}
double latOffset = 0.0;
if (strLatOffset != null) {
    latOffset = Double.parseDouble(strLatOffset);
}

String action = request.getParameter("action");
boolean forceTrace = false;
String retrieval;
if ("Force trace".equals(action)) {
    forceTrace = true;
    retrieval = "Forced new trace";
} else if ("Get trace".equals(action)) {
    if (TraceCache.hasTraceId(ip)) {
        retrieval = "Using cached trace";
    } else {
        retrieval = "Performed new trace";
    }
} else if ("Zoom in".equals(action)) {
    retrieval = "Refreshed map of previous trace";
} else if ("Zoom out".equals(action)) {
    retrieval = "Refreshed map of previous trace";
} else if ("Move up".equals(action)) {
    retrieval = "Refreshed map of previous trace";
} else if ("Move down".equals(action)) {
    retrieval = "Refreshed map of previous trace";
} else if ("Move left".equals(action)) {
    retrieval = "Refreshed map of previous trace";
} else if ("Move right".equals(action)) {
    retrieval = "Refreshed map of previous trace";
} else {
    sc.log("Unknown action");
    response.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
}
return;
}

if (ip == null) {
    sc.log("Destination not specified!");
    response.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
    return;
}

// Perform trace or retrieve cached trace
if (cacheId == null) {
    cacheId = TraceCache.getTraceId(ip, forceTrace);
}
TraceCacheEntry ce = TraceCache.getCachedTrace(cacheId, ip);
if (ce == null) {
    sc.log("Could not find cache id '" + cacheId + "' for address '" + ip + "'");
    response.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
    return;
}

IntersectionInfo intersectionInfo = MapDrawer.getIntersectionInfo(ce.cacheId, ce.circles);
//try { Thread.sleep(500); } catch (Exception e) {}
} else {
    out.println("longitude=" + degToString(intersectionInfo.
        centroidLongitude) + 
    ", latitude=" + degToString(intersectionInfo.
        centroidLatitude) + 
    "<br>");
}
out.println("<table><tr>");
out.println("<td><form action="Map" method=POST>");
out.println("<input type=hidden name=ip value=" + ip + ">");
out.println("<input type=hidden name=cacheId value=" + cacheId + ">");
out.println("<input type=hidden name=zoom value=" + (zoom * 2) + ">");
out.println("<input type=hidden name=longOffset value=" + (longOffset) + ">");
out.println("<input type=hidden name=latOffset value=" + (latOffset) + ">");
out.println("<input type=submit name=action value="Zoom in" ">");
out.println("</form></td>");
out.println("<td><form action="Map" method=POST>");
out.println("<input type=hidden name=ip value=" + ip + ">");
out.println("<input type=hidden name=cacheId value=" + cacheId + ">");
out.println("<input type=hidden name=zoom value=" + (zoom / 2) + ">");
out.println("<input type=hidden name=longOffset value=" + (longOffset) + ">");
out.println("<input type=hidden name=latOffset value=" + (latOffset) + ">");
out.println("<input type=submit name=action value="Zoom out" ">");
out.println("</form></td>");
out.println("<td><form action="Map" method=POST>");
out.println("<input type=hidden name=ip value=" + ip + ">");
out.println("<input type=hidden name=cacheId value=" + cacheId + ">");
out.println("<input type=hidden name=zoom value=" + (zoom) + ">");
out.println("<input type=hidden name=longOffset value=" + (longOffset) + ">");
out.println("<input type=hidden name=latOffset value=" + (latOffset+2) + ">");
out.println("<input type=submit name=action value="Move up" ">");
out.println("</form></td>");
out.println("<td><form action="Map" method=POST>");
out.println("<input type=hidden name=ip value=" + ip + ">");
out.println("<input type=hidden name=cacheId value=" + cacheId + ">");
out.println("<input type=hidden name=zoom value=" + (zoom) + ">");
out.println("<input type=hidden name=longOffset value=" + (longOffset) + ">");
out.println("<input type=hidden name=latOffset value=" + (latOffset+2) + ">");
out.println("<input type=submit name=action value="Move down" ">");
out.println("</form></td>");
out.println("<td><form action="Map" method=POST>");
out.println("<input type=hidden name=ip value=" + ip + ">");
out.println("<input type=hidden name=cacheId value=" + cacheId + ">");
out.println("<input type=hidden name=zoom value=" + (zoom) + ">");
out.println("<input type=hidden name=longOffset value=" + (longOffset) + ">");
out.println("<input type=hidden name=latOffset value=" + (latOffset+2) + ">");
out.println("<input type=submit name=action value="Move up" ">");
out.println("</form></td>");
out.println("<td><form action="Map" method=POST>");
out.println("<input type=hidden name=ip value=" + ip + ">");
out.println("<input type=hidden name=cacheId value=" + cacheId + ">");
out.println("<input type=hidden name=zoom value=" + (zoom) + ">");
out.println("<input type=hidden name=longOffset value=" + (longOffset) + ">");
out.println("<input type=hidden name=latOffset value=" + (latOffset+2) + ">");
out.println("<input type=submit name=action value="Move down" ">");
out.println("</form></td>");
out.print("<input type=hidden name=cacheId value=" + cacheId + " ">");
out.print("<input type=hidden name=zoom value=" + (zoom) + " ">");
out.print("<input type=hidden name=longOffset value=" + (longOffset) + " ">");
out.print("<input type=hidden name=latOffset value=" + (latOffset) + " ">");
out.println("<input type=submit name=action value="Move down">");
out.println("</form></td>");
out.println("<td><form action=Map\"\" method=POST">
out.print("<input type=hidden name=ip value=" + ip + " ">");
out.print("<input type=hidden name=cacheId value=" + cacheId + " ">");
out.print("<input type=hidden name=zoom value=" + (zoom) + " ">");
out.print("<input type=hidden name=longOffset value=" + (longOffset) + " ">");
out.print("<input type=hidden name=latOffset value=" + (latOffset) + " ">");
out.println("<input type=submit name=action value="Move left">");
out.println("</form></td>");
out.println("<td><form action=Map\"\" method=POST">
out.print("<input type=hidden name=ip value=" + ip + " ">");
out.print("<input type=hidden name=cacheId value=" + cacheId + " ">");
out.print("<input type=hidden name=zoom value=" + (zoom) + " ">");
out.print("<input type=hidden name=longOffset value=" + (longOffset) + " ">");
out.print("<input type=hidden name=latOffset value=" + (latOffset) + " ">");
out.println("<input type=submit name=action value="Move right">");
out.println("</form></td>");
out.println("</tr></table>");
out.println("<img src="MapImage?" + "ip=" + ip + 
"&cacheId=" + cacheId + 
"&zoom=" + zoom + 
"&longOffset=" + longOffset + 
"&latOffset=" + latOffset + 
" "+ 
"border=1" + 
"width=" + intersectionInfo.imageWidth + 
"height=" + intersectionInfo.imageHeight + 
"alt="OpenMap(tm)image" <br/>");
out.println("</body> ");
out.println("</html> ");
@Override
public void doPost(HttpServletRequest request,
        HttpServletResponse response)
        throws IOException, ServletException
{
    doGet(request, response);
}

private String degToString(double degrees) {
    int deg = (int)degrees;
    double minutes = (degrees - deg) * 60;
    ByteArrayOutputStream tmp = new ByteArrayOutputStream();
    PrintStream stream = new PrintStream(tmp);
    stream.printf(Locale.US, "%d° %.2f', deg, minutes);
    return tmp.toString();
}
C.6.3 GetMapImage

```java
package kripos.math.servlet;

import java.io.IOException;
import java.io.OutputStream;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import kripos.geo.openmap.MapDrawer;

/**
 * Create a map image from the cached circles corresponding to
 * the given ID, using OpenMap.
 * @author oysteine
 */
public class GetMapImage extends HttpServlet {

    private static final long serialVersionUID = 6720480250134837969L;

    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws IOException, ServletException {
        ServletContext sc = getServletContext();
        String ip = request.getParameter("ip");
        String cacheId = request.getParameter("cacheId");
        TraceCacheEntry ce = TraceCache.getCachedTrace(cacheId, ip);
        if (ce == null) {
            sc.log("Could not find cache id '" + cacheId + "' for address '" + ip + ",");
            response.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
            return;
        }

        String strZoom = request.getParameter("zoom");
        String strLongOffset = request.getParameter("longOffset");
        String strLatOffset = request.getParameter("latOffset");
        float zoom = 1.0f;
        if (strZoom != null) {
```
double longOffset = 0.0;
if (strLongOffset != null) {
    longOffset = Double.parseDouble(strLongOffset);
}

double latOffset = 0.0;
if (strLatOffset != null) {
    latOffset = Double.parseDouble(strLatOffset);
}

byte[] image = getImage(ce, zoom, longOffset, latOffset);
String filename = "mapView.gif";
String mimeType = sc.getMimeType(filename);
if (mimeType == null) {
    sc.log("Could not get MIME type of " + filename);
    response.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
    return;
}

response.setContentType(mimeType);
response.setContentLength(image.length);
OutputStream out = response.getOutputStream();
out.write(image);
out.close();

@Override
public void doPost(HttpServletRequest request,
        HttpServletResponse response)
throws IOException, ServletException
{
    doGet(request, response);
}

private byte[] getImage(TraceCacheEntry ce,
    float zoom, double longOffset, double latOffset) {
    return MapDrawer.getMap(ce.cacheId, ce.circles,
    zoom, latOffset, longOffset);
}
C.6.4 TraceCache

```java
package kripos.math.servlet;

import jade.util.BasicProperties;
import jade.wrapper.ControllerException;
import jade.wrapper.StaleProxyException;
import jade.wrapper.gateway.JadeGateway;
import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.Random;
import kripos.gateway.CommandPackage;
import kripos.math.circle.Circle;
import kripos.ontology.GeoLocation;
import kripos.ontology.TraceResult;

/**< *
* Perform IP tracing and add the results to cache.
* @author oysteine
**/
public class TraceCache {

/**< */
/** Max life time of a cache entry in milliseconds. */
static final long CACHE_LIFE_TIME = 24 * 3600 * 1000L;

static private final Random rand = new Random();

/**
* Mapping from cacheId. *
private final static Map<Long,TraceCacheEntry> idMap =
new HashMap<Long,TraceCacheEntry>();

/**< */
/** Mapping from IP to list of cacheId for all results
* performed for within cache life time.
* Most recent trace is stored first in the list. */
private final static Map<String,List<Long>> destinationMap =
new HashMap<String,List<Long>>();

/**< */
* Check if <code>destination</code> is cached.
* 
```
```
* @param ip
* @return <code>true</code> iff <code>destination</code> can
* be retrieved with {@link #getTraceId(String, boolean)}
* without performing a trace
*
*/
static boolean hasTraceId(String destination) {
    synchronized (cacheMutex) {
        List<Long> tmp = destinationMap.get(destination);
        if (tmp != null) {
            Long id = tmp.get(0);
            TraceCacheEntry ce = idMap.get(id);
            ce.lastAccess = new Date(); // to make sure it’s still
              available when retrieving it a little later
            return true;
        } else {
            return false;
        }
    }
}

/**
 * Perform a trace or return most recent previous trace of <
 * code>ip</code>,
 * if one has been done.
 * @param ip a valid IP address
 * @param forceNew if <code>true</code> a new trace will
 * always be performed,
 * even if <code>ip</code> already exists in the
 * cache
 * @return a cache id that can be used with {@link #
 * getCachedTrace(String, String)}
 */
static String getTraceId(String destination, boolean forceNew) {
    cleanCache();
    Long id = null;
    if (!forceNew) {
        synchronized (cacheMutex) {
            List<Long> tmp = destinationMap.get(destination);
            if (tmp != null) {
                id = tmp.get(0);
            }
        }
    }
    if (id == null) {
        // NOTE: it is possible that two traces are performed
        // with the same IP
        // at the same time
        TraceCacheEntry ce = performTrace(destination);
        insertToCache(ce);
        id = ce.cacheId;
/**
 * Retrieve the trace specified by `<code>cacheId</code>` from cache.
 * @param cacheId an id retrieved by {@link #getTrace(String, boolean)} recently
 * @param destination used to assert that the `<code>cacheId</code>` entry was performed with this destination.
 * If it was not, the request is treated as if `<code>cacheId</code>` was not found,
 * it does not exist, or
 * it does not match `<code>destination</code>`
 * @return `<code>null</code>` if `<code>cacheId</code>` is not malformed,
 */
static TraceCacheEntry getCachedTrace(String cacheId, String destination) {
    cleanCache();

    Long id = null;
    try {
        id = Long.parseLong(cacheId);
    } catch (NumberFormatException e) {
        return null;
    }

    TraceCacheEntry ce;
    synchronized (cacheMutex) {
        ce = idMap.get(id);
    }  

    if (ce == null || !ce.destination.equals(destination)) {
        return null;
    }

    return ce;
}

static private void insertToCache(TraceCacheEntry ce) {
    synchronized (cacheMutex) {
        long id = -1;
        synchronized (rand) {
            while (id < 0 || idMap.containsKey(id)) {
                id = rand.nextLong();
            }
        }
    }
}
ce.cacheId = id;

idMap.put(ce.cacheId, ce);

List<Long> tmp = destinationMap.get(ce.destination);
if (tmp == null) {
    tmp = new LinkedList<Long>();
    destinationMap.put(ce.destination, tmp);
}
    tmp.add(0, ce.cacheId);
}

/**
 * Remove entries from cache if they are older than
 * {@linkplain #CACHE_LIFE_TIME} ms.
 */
static private void cleanCache() {
    synchronized (cacheMutex) {
        long curTime = System.currentTimeMillis();
        for (Map.Entry<String,List<Long>> entry : destinationMap.
            entrySet()) {
            List<Long> idList = entry.getValue();
            Iterator<Long> iter = idList.iterator();
            while (iter.hasNext()) {
                Long id = iter.next();
                TraceCacheEntry ce = idMap.get(id);
                if (curTime - ce.lastAccess.getTime() > CACHE_LIFE_TIME) {
                    idMap.remove(id);
                    iter.remove();
                }
            }
            if (idList.size() == 0) {
                destinationMap.remove(entry.getKey());
            }
        }
    }

    /**
     * @param destination a unique destination identifier
     * @return resulting circles from a trace
     */
    static private TraceCacheEntry performTrace(String destination) {
        long startTime = System.currentTimeMillis();
        List<Circle> circles = TRACELIBRARY_getTrace(destination);
        long endTime = System.currentTimeMillis();
        return new TraceCacheEntry(}
destination, circles,
    new Date(startTime), endTime - startTime);
}

/**
 * Interfaces with the multi-agent system to perform a trace
 * @param the IP address to trace
 * @return the geographical constraint circles
 */
static private List<Circle> TRACELIBRARY_getTrace(String destination) {
    double KMperDegree = 1/111.15;
    List<Circle> circles = new LinkedList<Circle>();

    BasicProperties prop = new BasicProperties();
    prop.setProperty("platform-id", "futurum01.item.ntnu.no:1099/JADE"); // hard coded for now :
    prop.setBooleanProperty("main", true);
    prop.setProperty("mainURL","http://futurum01.item.ntnu.no:7778/acc"); // hard coded for now :
    JadeGateway.init("kripos.gateway.GWAgent", prop);
    CommandPackage cp = new CommandPackage();
    cp.setTarget(destination);
    cp.setType("TRACE-CBG");

    try {
        JadeGateway.execute(cp);
    } catch (StaleProxyException e) {
        e.printStackTrace();
    } catch (ControllerException e) {
        e.printStackTrace();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    ArrayList<TraceResult> traceRes = cp.getCBGResults();
    for(TraceResult tr : traceRes){
       GeoLocation geoLoc = tr.getHasGeoLocation();
        double lat = geoLoc.getLocationLatitude();
        double lon = geoLoc.getLocationLongitude();
        double radius = tr.getTraceResultData() * KMperDegree;
        circles.add(new Circle(lat, lon, radius));
    }

    return circles;
}
} // class
C.6.5 TraceCacheEntry

```java
package kripos.math.servlet;

import java.util.Date;
import java.util.List;
import kripos.math.circle.Circle;

class TraceCacheEntry {

    /** Address that was traced. */
    final String destination;

    /** Result of trace. */
    final List<Circle> circles;

    /** Time trace was started. */
    final Date start;

    /** Milliseconds used to perform the trace. */
    final long runTime;

    /** Last time this cache entry was used. */
    Date lastAccess;

    /**
     * Internal {@link TraceCache} cache id.
     * Set before added to the cache.
     */
    long cacheId = -1;

    /**
     * You must set {@link #cacheId} before inserting this entry to cache
     *
     * @param destination
     * @param circles
     */
    TraceCacheEntry(
            String destination, List<Circle> circles,
            Date start, long runTime) {
        this.destination = destination;
        this.circles = circles;
        this.start = start;
        this.runTime = runTime;
        this.lastAccess = new Date();
    }
}
```
C.6.6 MapDrawer

```java
package kripos.geo.openmap;

import java.awt.Color;
import java.awt.Point;
import java.awt.Shape;
import java.awt.geom.Area;
import java.awt.geom.FlatteningPathIterator;
import java.awt.geom.GeneralPath;
import java.awt.geom.PathIterator;
import java.awt.geom.Point2D;
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.Properties;
import java.util.concurrent.locks.ReentrantLock;

import kripos.math.circle.Circle;
import kripos.math.circle.Intersection;
import kripos.math.circle.Intersector;
import kripos.math.servlet.GetMap;

import com.bbn.openmap.LatLonPoint;
import com.bbn.openmap.Layer;
import com.bbn.openmap.MapBean;
import com.bbn.openmap.event.LayerStatusEvent;
import com.bbn.openmap.event.LayerStatusListener;
import com.bbn.openmap.image.AcmeGifFormatter;
import com.bbn.openmap.layer.OMGraphicHandlerLayer;
import com.bbn.openmap.layer.location.LocationHandler;
import com.bbn.openmap.layer.location.csv.CSVLocationHandler;
import com.bbn.openmap.layer.shape.ShapeLayer;
import com.bbn.openmap.omGraphics.OMCircle;
import com.bbn.openmap.omGraphics.OMGraphic;
import com.bbn.openmap.omGraphics.OMGraphicList;
import com.bbn.openmap.omGraphics.OMPoint;
import com.bbn.openmap.omGraphics.OMPoly;
import com.bbn.openmap.proj.Length;
import com.bbn.openmap.proj.Mercator;
import com.bbn.openmap.proj.Projection;

/**<n
 * Draw circles on a map.
 * @author oysteine
 */
public class MapDrawer {

    private static final int MAX_CACHE = 100;
```
private static final Object cacheMutex = new Object();
private static final Map<Long, CacheEntry> mapCache =
    new HashMap<Long, CacheEntry>();
private static final List<Long> cacheLru = new LinkedList<Long>();

private static class CacheEntry {
    private final Long cacheId;
    private final ReentrantLock lock = new ReentrantLock();
    private MapBean mapBean = null;
    private LatLonPoint pointUpperLeft;
    private LatLonPoint pointLowerRight;
    private LatLonPoint pointCenter;
    private float scale;
    private float origScale;
    private double area;
    private LatLonPoint centroid = null;
    private LayerListener listener;
    private LocationHandler locationHandler;

    private CacheEntry(Long cacheId) {
        this.cacheId = cacheId;
    }
}

private static CacheEntry acquireMap(Long cacheId) {
    CacheEntry ce;
    synchronized (cacheMutex) {
        ce = mapCache.get(cacheId);
        if (ce == null) {
            ce = unsafeNewCacheEntry(cacheId);
        }
        cacheLru.remove(cacheId);
        cacheLru.add(cacheId);
    }
    ce.lock.lock();
    return ce;
}

private static void releaseMap(CacheEntry ce) {
    synchronized (cacheMutex) {
        ce.lock.unlock();
        cacheLru.add(ce.cacheId);
    }
}

private static CacheEntry unsafeNewCacheEntry(Long cacheId) {

assert Thread.holdsLock(cacheMutex);

CacheEntry ce = new CacheEntry(cacheId);
mapCache.put(cacheId, ce);
if (mapCache.size() > MAX_CACHE) {
  mapCache.remove(cacheLru.remove(0));
}
return ce;

/**
 * @param cacheId use cached drawing instead of creating using <code>circles</code>,
 * if the cache holds anything
 * @param circles
 * @return informat about area and centroid that will be drawn on map
 */
static public IntersectionInfo getIntersectionInfo(
  Long cacheId, List<Circle> circles) {
  CacheEntry ce = acquireMap(cacheId);
  try {
    if (ce.mapBean == null) {
      createMap(ce, circles);
    }
    if (ce.centroid == null) {
      return new IntersectionInfo(
        ce.area,
        ce.mapBean.getWidth(),
        ce.mapBean.getHeight());
    } else {
      return new IntersectionInfo(
        ce.area,
        ce.centroid.getLatitude(),
        ce.centroid.getLongitude(),
        ce.mapBean.getWidth(),
        ce.mapBean.getHeight());
    }
  } finally {
    releaseMap(ce);
  }
}

private static final Proj dummyProj =
  new Mercator(new LatLonPoint(0.123456, -0.123456), 1.42E7f, 640, 480);
/**
 * @param cacheId use cached drawing instead of creating using <code>circles</code>,
 * if the cache holds anything
 * @param circles
 * @param zoom 1.0 is with polygon extremes on the borders,
 * smaller values are larger portion of the world,
 * greater values are zoomed inside polygon
 * @param latOffset north offset of quasi center of polygon
 * in decimal degrees
 * @param longOffset east offset of quasi center of polygon
 * in decimal degrees
 * @return a map with <code>circles</code> painted on it
 */
static public byte[] getMap(Long cacheId, List<Circle> circles, float zoom, double latOffset, double longOffset) {
    CacheEntry ce = acquireMap(cacheId);
    try {
        if (ce.mapBean == null) {
            createMap(ce, circles);
        }
        LatLonPoint customCenter = new LatLonPoint(
                ce.pointCenter.getLatitude() + latOffset,
                ce.pointCenter.getLongitude() + longOffset);
        float customScale = ce.scale / zoom;
        if (customScale > ce.origScale) {
            customScale = ce.origScale;
        }
        // disable city names if showing too much of world at once
        ce.locationHandler.setShowNames(customScale < 2.5E7);
        // update map
        Proj proj = new Mercator(customCenter, customScale,
                ce.mapBean.getWidth(), ce.mapBean.getHeight());
        // if we use only default parameters the first time, then
        // for some reason:
        // - traceLayer is not drawn
        // - repaints hang in LayerListener
        // so we use a dummy projection first
        ce.listener.resetCompletion(1);
        ce.mapBean.setProjection(dummyProj);
        ce.listener.waitForCompletion();
    } else {
        ce.listener.resetCompletion(1);
    }
}
ce.mapBean.setProjection(proj);
ce.listener.waitForCompletion();

// create image
AcmeGifFormatter gifFormatter = new AcmeGifFormatter();
byte[] image = gifFormatter.getImageFromMapBean(ce.
    mapBean);
return image;

} finally {
    releaseMap(ce);
}

private static void createMap(CacheEntry ce, List<Circle>
circles) {
    String shapeFile = GetMap.getDataPath() + "/shape/dcwpo-
browse.shp";
    String spatialFile = GetMap.getDataPath() + "/shape/dcwpo-
browse.ssx";
    String locationFile = GetMap.getDataPath() + "/cities.csv";
    ce.listener = new LayerListener();
    MapBean mapBean = new MapBean();
    mapBean.setSize(1024, 768);
    mapBean.suppressCopyright = true; // suppress after first
    use
    // political borders
    Shapelayer backgroundLayer = new Shapelayer();
    ce.listener.addLayer(backgroundLayer, 1);
    Properties backgroundProps = new Properties();
    backgroundProps.put("prettyName", "Political\nSolid");
    backgroundProps.put("lineColor", "000000");
    backgroundProps.put("fillColor", "BDDE83");
    backgroundProps.put("shapeFile", shapeFile);
    backgroundProps.put("spatialIndex", spatialFile);
    backgroundLayer.setProperties(backgroundProps);
    assert mapBean.getComponentCount() == 0 : mapBean.
    getComponentCount();
    mapBean.add(backgroundLayer, 0);
    OMGraphicList traceItems = new OMGraphicList(2);
    OMGraphicList tracePolygon = new OMGraphicList(1);
    OMGraphicList traceCircles = new OMGraphicList(circles.size
    ());
    // inaccurate intersections used for initial scale
    calculation
    List<Intersection> inters2D = Intersector.getIntersections(
circles);
inters2D = Intersector.getMorePoints(inters2D, 0.2);

setBorders(ce, inters2D, traceItems, mapBean);
setBorderInfo(ce, traceItems);

setPolygon2D(inters2D, tracePolygon);
setCircles(circles, traceCircles);

OMGraphicHandlerLayer traceLayer = new OMRGraphicHandlerLayer();
ce.listener.addLayer(traceLayer, 1);
traceItems.add(tracePolygon);
traceItems.add(traceCircles);
traceLayer.setList(traceItems);
mapBean.add(traceLayer, 0);
ce.listener.waitForCompletion();

List<Point2D> screenPath = getWindingPath(ce, traceCircles, mapBean);
Point centroidXY = getScreenCentroid(screenPath);
if (centroidXY == null) {
    ce.centroid = null;
} else {
    ce.centroid = mapBean.getProjection().inverse(centroidXY);
}

List<LatLonPoint> approximatedPath = new LinkedList<LatLonPoint>();
for (Point2D next : screenPath){
    Point temp = new Point((int)next.getX(), (int)next.getY());
    approximatedPath.add(mapBean.getProjection().inverse(temp));
}
ce.area = getApproximatedPolygonArea(approximatedPath);

traceItems = new OMRGraphicList(2);
tracePolygon = new OMRGraphicList(1);
traceCircles = new OMRGraphicList(circles.size());

setBorderInfo(ce, traceItems);
setApproximatedPolygon(approximatedPath, tracePolygon);
setCircles(circles, traceCircles);

LocationLayer locations = new LocationLayer();
ce.locationHandler = new CSVLocationHandler();
Properties lhProps = new Properties();
lhProps.put("locationFile", locationFile);
lhProps.put("csvFileHasHeader", "false");
lhProps.put("showNames", "true");
//lhProps.put("nameColor", "008C54");
lhProps.put("nameColor", "000000");
lhProps.put("showLocations", "true");
lhProps.put("locationColor", "FF0000");
lhProps.put("name.lineColor", "FF008C54");
lhProps.put("location.lineColor", "FFFF0000");
lhProps.put("location.fillColor", "FfFfAaaaAa");
lhProps.put("location.pointRadius", "2");
lhProps.put("location.pointOval", "true");
lhProps.put("nameIndex", "0");
lhProps.put("latIndex", "2");
lhProps.put("lonIndex", "3");
ce.locationHandler.setProperties(lhProps);

//locations.setProperties(lhProps);
locations.setLocationHandlers(new LocationHandler[]{ce.locationHandler});

Proj proj = new Mercator(ce.pointCenter, ce.scale,
    mapBean.getWidth(), mapBean.getHeight());
ce.listener.resetCompletion(1);
traceItems.add(tracePolygon);
traceItems.add(traceCircles);
traceLayer.setList(traceItems);
mapBean.add(locations, 0);
mapBean.setProjection(proj);
ce.listener.waitForCompletion();

ce.mapBean = mapBean;
}

private static double getApproximatedPolygonArea(List<
    LatLonPoint> sortedPath){
    if (sortedPath.size() < 3) {
        return 0.0;
    }
    List<LatLonPoint> tmp = new LinkedList<LatLonPoint>(
        sortedPath);
    LatLonPoint base = tmp.remove(0);
    int triangleCount = 0;
    double interiorAngleSum = 0.0;
    LatLonPoint prev = tmp.remove(0);
    for (LatLonPoint next : tmp) {
        LatLonPoint A = base;
        LatLonPoint B = prev;
        LatLonPoint C = next;
        double a = B.distance(C);
        double b = C.distance(A);
        double c = A.distance(B);
        if (a >= Math.ulp(a) && b >= Math.ulp(b) && c >= Math.ulp(c)) {
            double angleA = Math.acos(
                ...
```
            (Math.cos(a) - Math.cos(b) * Math.cos(c)) / 
            (Math.sin(b) * Math.sin(c));
        double angleB = Math.acos(
            (Math.cos(b) - Math.cos(c) * Math.cos(a)) / 
            (Math.sin(c) * Math.sin(a)));
        double angleC = Math.acos(
            (Math.cos(c) - Math.cos(b) * Math.cos(a)) / 
            (Math.sin(a) * Math.sin(b)));
        interiorAngleSum += angleA + angleB + angleC;
        triangleCount++;
    } else {
        // 1 distance is 0, the other two are equal, OR
        // all distances are 0
        //assert a < Math.ulp(a) : A + "," + B + "," + C;
        //assert b < Math.ulp(b) : A + "," + B + "," + C;
        //assert c < Math.ulp(c) : A + "," + B + "," + C;
    }
}
prev = next;
}
// approx, should use rad at current geodetic latitude
double earthRad = (6378.135 + 6356.750) / 2.0;
double area =
    Math.pow(earthRad, 2) *
    (interiorAngleSum - Math.PI * triangleCount);
return area;
}
private static double getScreenPolygonArea(List<Point2D>
    sortedPath){
    if (sortedPath.size() == 0) {
        return 0.0;
    }
    double area = 0.0;
    List<Point2D> tmp = new LinkedList<Point2D>(sortedPath);
    Point2D prev = tmp.remove(0);
    tmp.add(prev);
    for (Point2D next : tmp) {
        double x1 = prev.getX();
        double y1 = prev.getY();
        double x2 = next.getX();
        double y2 = next.getY();
        area += x1*y2 - x2*y1;
        prev = next;
    }
    area *= 0.5;
    return area;
}
private static Point getScreenCentroid(List<Point2D> points)
```
if (points.size() < 2) {
    return null;
}

double area = getScreenPolygonArea(points);

double x = 0.0;
double y = 0.0;

List<Point2D> tmp = new LinkedList<Point2D>(points);
Point2D prev = tmp.remove(0);
tmp.add(prev);

for (Point2D next : tmp) {
    double x1 = prev.getX();
    double y1 = prev.getY();
    double x2 = next.getX();
    double y2 = next.getY();

    double determinant = x1*y2 - x2*y1;
    x += (x1+x2) * determinant;
    y += (y1+y2) * determinant;

    prev = next;
}

x /= (6 * area);
y /= (6 * area);

return new Point((int)x,(int)y);

private static List<Point2D> getWindingPath(
    CacheEntry ce, OMGraphicList circles,
    MapBean mapBean)
{
    List<Point2D> ret = new LinkedList<Point2D>();

    if (circles.size() == 0) {
        return ret;
    }

    Proj proj = new Mercator(ce.pointCenter, ce.scale,
        mapBean.getWidth(), mapBean.getHeight());
    ce.listener.resetCompletion(1);
    mapBean.setProjection(proj);
    ce.listener.waitForCompletion();

    Iterator iter = circles.iterator();
    OMCircle circle = (OMCircle)iter.next();

    Shape s = circle.getShape();
    assert s != null;
    Area polygon = new Area(s);
while (iter.hasNext()) {
    circle = (OMCircle) iter.next();
    s = circle.getShape();
    assert s != null;
    Area area = new Area(s);
    polygon.intersect(area);
}

PathIterator pathIter = polygon.getPathIterator(null);
GeneralPath genPath = new GeneralPath();
int theType;
float[] theData = new float[6];
while (!pathIter.isDone()) {
    theType = pathIter.currentSegment(theData);
    switch (theType) {
    case PathIterator.SEG_MOVETO :
        genPath.moveTo(theData[0], theData[1]);
        break;
    case PathIterator.SEG_LINETO :
        genPath.lineTo(theData[0], theData[1]);
        break;
    case PathIterator.SEG_QUADTO :
        genPath.quadTo(theData[0], theData[1], theData[2],
                       theData[3]);
        break;
    case PathIterator.SEG_CUBICTO :
        genPath.curveTo(theData[0], theData[1], theData[2],
                        theData[3], theData[4], theData[5]);
        break;
    case PathIterator.SEG_CLOSE :
        genPath.closePath();
        break;
    } // end switch
    pathIter.next();
}

PathIterator pathIter2 = genPath.getPathIterator(null);
FlatteningPathIterator fpi = new FlatteningPathIterator(pathIter2, 0.25);
double[] coords = new double[6];
while (!fpi.isDone()) {
    fpi.currentSegment(coords);
    ret.add(new Point2D.Double(coords[0], coords[1]));
    fpi.next();
}
return ret;
private static void setApproximatedPolygon(List<LatLonPoint> approximatePath, OMGraphicList traceItems) {
    if (approximatePath.size() > 0) {
        float[] lats_lons = new float[2*(approximatePath.size() + 1)];
        int i = 0;
        for (LatLonPoint point : approximatePath) {
            lats_lons[i++] = (float) point.getLatitude();
            lats_lons[i++] = (float) point.getLongitude();
        }
        LatLonPoint firstPoint = approximatePath.get(0);
        lats_lons[i++] = (float) firstPoint.getLatitude();
        lats_lons[i++] = (float) firstPoint.getLongitude();
        OMPoly omPolygon = new OMAAlphaPoly(lats_lons, OMGraphic.DECIMAL_DEGREES,
                OMGraphic.LINE_TYPE_STRAIGHT, 0.1f);
        omPolygon.setLinePaint(Color.black);
        omPolygon.setFillPaint(Color.red);
        traceItems.add(omPolygon);
    }
}

private static void setBorders(CacheEntry ce, List<Intersection> inters, OMGraphicList traceItems, MapBean mapBean) {
    double northMost = Double.MIN_VALUE;
    double southMost = Double.MAX_VALUE;
    double westMost = Double.MAX_VALUE;
    double eastMost = Double.MIN_VALUE;
    if (inters.size() == 0) {
        northMost = 90.0;
        southMost = -90.0;
        westMost = -180.0;
        eastMost = 180.0;
    } else {
        for (Intersection inter : inters) {
            double longitude = inter.point.getY();
            double latitude = inter.point.getX();
            if (latitude > northMost) {
                northMost = latitude;
            } else {
                if (latitude < southMost) {
                    southMost = latitude;
                } else {
                    if (longitude < westMost) {
                        westMost = longitude;
                    } else {
                        if (longitude > eastMost) {
                            eastMost = longitude;
                        }
                    }
                }
            }
        }
    }
}
ce.pointUpperLeft = new LatLonPoint(northMost, westMost);
ce.pointLowerRight = new LatLonPoint(southMost, eastMost);

Projection prevProj = mapBean.getProjection();
Point pUL = prevProj.forward(ce.pointUpperLeft);
Point pLR = prevProj.forward(ce.pointLowerRight);
ce.origScale = prevProj.getScale();

ce.scale = 2f * prevProj.getScale(
    ce.pointUpperLeft, ce.pointLowerRight, pUL, pLR);

ce.pointCenter = new LatLonPoint(
    northMost - (northMost - southMost) / 2,
    westMost  + (eastMost - westMost) / 2);
}

private static void setBorderInfo(
    CacheEntry ce, OMGraphicList traceItems) {
    if (ce.centroid != null) {
        OMPoint p = new OMPoint(
            ce.centroid.getLatitude(),
            ce.centroid.getLongitude(),
            5);
        p.setFillPaint(Color.pink);
        traceItems.add(p);
    }
}

private static void setPolygon2D(List<Intersection> inters,
    OMGraphicList traceItems) {
    if (inters.size() > 0) {
        float[] lats_lons = new float[2*(inters.size()+1)];
        int i = 0;
        for (Intersection inter : inters) {
            lats_lons[i++] = (float)inter.point.getX();
            lats_lons[i++] = (float)inter.point.getY();
        }
        Intersection firstInter = inters.get(0);
        lats_lons[i++] = (float)firstInter.point.getX();
        lats_lons[i++] = (float)firstInter.point.getY();
        OMPoly omPolygon = new OMAlphaPoly(lats_lons, OMGraphic.
            DECIMAL_DEGREES,
            OMGraphic.LINE_TYPE_STRAIGHT, 0.4f);
        omPolygon.setLinePaint(Color.black);
        omPolygon.setFillPaint(Color.red);
        traceItems.add(omPolygon);
    }
}
private static void setCircles(List<Circle> circles, OMGGraphicList traceItems) {
    for (Circle circle : circles) {
        Point2D p = circle.origo;
        LatLonPoint center = new LatLonPoint(p.getX(), p.getY());
        OMCircle omCircle = new OMAICircle(center, (float)circle.rad, Length.DECIMAL_DEGREE, 0, 0.2f);
        omCircle.setLinePaint(Color.black);
        omCircle.setFillPaint(Color.black);
        traceItems.add(omCircle);
    }
}

private static class LayerListener implements LayerStatusListener {
    private final Object completionMutex = new Object();
    private final Map<Layer, Integer> completionMap = new HashMap<Layer, Integer>();

    private void addLayer(Layer layer, int remainingCount) {
        synchronized (completionMutex) {
            layer.addLayerStatusListener(this);
            completionMap.put(layer, remainingCount);
        }
    }

    public void updateLayerStatus(LayerStatusEvent evt) {
        synchronized (completionMutex) {
            switch (evt.getStatus()) {
            case LayerStatusEvent.DISTRESS:
                break;
            case LayerStatusEvent.FINISH_WORKING:
                completionMap.put(evt.getLayer(), completionMap.get(evt.getLayer()) - 1);
                completionMutex.notifyAll();
                break;
            case LayerStatusEvent.START_WORKING:
                break;
            case LayerStatusEvent.STATUS_UPDATE:
                break;
            }
        }
    }

    private void resetCompletion(int remainingCount) {
        synchronized (completionMutex) {
            for (Layer layer : completionMap.keySet()) {
                completionMap.put(layer, remainingCount);
            }
        }
    }
}
completionMap.put(layer, remainingCount);
}
}
}

private void waitForCompletion() {
    synchronized (completionMutex) {
        boolean allDone = true;
        for (Integer remaining: completionMap.values()) {
            if (remaining > 0) {
                allDone = false;
                break;
            }
        }
        if (allDone) {
            break;
        }
        try {
            long startTime = System.currentTimeMillis();
            completionMutex.wait(5000);
            long endTime = System.currentTimeMillis();
            if (endTime - startTime > 4500) {
                break;
            }
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }
}  // class
C.6.7 IntersectionInfo

```java
package kripos.geo.openmap;

/**
 * Information about the intersection represented in a map.
 * @author oysteine
 */
public class IntersectionInfo {

    /**
     * Calculated area of intersection polygon.
     */
    public final double polygonArea;

    /**
     * Whether {@link #centroidLatitude} and {@link #centroidLongitude}
     * contains legal values.
     */
    public final boolean centroidAvailable;

    /**
     * Latitude (north of Equator) of polygon centroid.
     */
    public final double centroidLatitude;

    /**
     * Longitude (east of Greenwich) of polygon centroid.
     */
    public final double centroidLongitude;

    /**
     * Pixel size of image.
     */
    public final int imageWidth;

    /**
     * Pixel size of image.
     */
    public final int imageHeight;

    IntersectionInfo(double polygonArea, int imageWidth, int imageHeight) {
        this.polygonArea = polygonArea;
        this.centroidAvailable = false;
        this.centroidLatitude = Double.NaN;
        this.centroidLongitude = Double.NaN;
        this.imageWidth = imageWidth;
        this.imageHeight = imageHeight;
    }
}
```
IntersectionInfo(double polygonArea,
    double centroidLatitude, double centroidLongitude,
    int imageWidth, int imageHeight) {
    this.polygonArea = polygonArea;
    this.centroidAvailable = true;
    this.centroidLatitude = centroidLatitude;
    this.centroidLongitude = centroidLongitude;

    this.imageWidth = imageWidth;
    this.imageHeight = imageHeight;
}

} // class
C.6.8 Alpha

```java
package kripos.geo.openmap;

/**
 * Implemented by OMGraphic subclasses that support
 * semi-transparent fill.
 * @author oysteine
 * @version 1.0
 */
public interface Alpha {
    /**
     * @param alpha opacity in [0.0, 1.0]
     */
    public void setAlpha(float alpha);
}
```
C.6.9 OMAlphaCircle

```java
package kripos.geo.openmap;
import java.awt.AlphaComposite;
import java.awt.Composite;
import java.awt.Graphics;
import java.awt.Graphics2D;
import com.bbn.openmap.LatLonPoint;
import com.bbn.openmap.omGraphics.OMCircle;
import com.bbn.openmap.proj.Length;

/**
 * An {@link OMCircle} implementation that supports transparency.
 * Only the directly used constructors are implemented
 * @author oysteine
 */
public class OMAlphaCircle extends OMCircle implements Alpha {
    private static final long serialVersionUID = 9141255017478768485L;
    private Composite composite;

    /**
     * Create an OMCircle with a lat/lon center and a physical distance radius. Rendertype is RENDERTYPE_LATLON.
     * @param center LatLon center of circle
     * @param radius distance
     * @param units com.bbn.openmap.proj.Length object specifying units for distance.
     * @param nverts number of vertices for the poly-circle(if &lt; 3, value is generated internally)
     * @param alpha opacity in [0.0, 1.0]
     */
    public OMAlphaCircle(LatLonPoint center, float radius, Length units,
                               int nverts, float alpha) {
        super(center, radius, units, nverts);
        composite = AlphaComposite.getInstance(AlphaComposite.SRC_OVER, alpha);
    }

    /**
     * Set the alpha color
     * @param the color value to set
     */
    public void setAlpha(int alpha) {
        composite = AlphaComposite.getInstance(AlphaComposite.SRC_OVER, alpha);
    }
}
```
```java
public void setAlpha(float alpha) {
    composite = AlphaComposite.getInstance(AlphaComposite.
    SRC_OVER, alpha);
}

/**
 * Overriding the fill method to set alpha before filling and
 * clearing it after.
 * @param g the <code>Graphics</code> instance to use
 */
@Override
public void fill(Graphics g) {
    Graphics2D g2 = (Graphics2D)g;
    Composite orig = g2.getComposite();
    g2.setComposite(composite);
    super.fill(g);
    g2.setComposite(orig);
}
//class
```
package kripos.geo.openmap;

import java.awt.AlphaComposite;
import java.awt.Composite;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Paint;

import com.bbn.openmap.omGraphics.OMPoly;

/**<n *
* An `{@link OMPoly}` implementation that supports transparency.
* *
* Only the directly used constructors are implemented
* *
* @author oysteine
*/
public class OMAlphaPoly extends OMPoly implements Alpha {
    private static final long serialVersionUID = 597512041926004097L;
    private Composite composite;
    private Composite orig;

    /**<n *
    * @param llPoints latitude, longitude, latitude, longitude, ...
    * @param units
    * @param lType
    * @param alpha opacity in [0.0, 1.0]
    */
    public OMAlphaPoly(float[] llPoints, int units, int lType, float alpha) {
        super(llPoints, units, lType);
        composite = AlphaComposite.getInstance(AlphaComposite.SRC_OVER, alpha);
    }

    /**<n *
    * Set the alpha color
    * *
    * @param the color value to set
    */
    public void setAlpha(float alpha) {
        composite = AlphaComposite.getInstance(AlphaComposite.SRC_OVER, alpha);
    }

    @Override
}
public void render(Graphics g) {
    // just to make sure we reset composite after
    Graphics2D g2 = (Graphics2D)g;
    orig = g2.getComposite();
    super.render(g);
    g2.setComposite(orig);
}

@Override
public void setGraphicsForFill(Graphics g) {
    ((Graphics2D)g).setComposite(composite);
    super.setGraphicsForFill(g);
}

@Override
public void setGraphicsColor(Graphics g, Paint paint) {
    ((Graphics2D)g).setComposite(composite);
    super.setGraphicsColor(g, paint);
}

} // class
C.6.11 Circle

//legacy code. only works in 2D. originally meant used with a flat map based on UTM coordinates

//turned out to be not accurate enough when spanning multiple UTM zones

//only used for setting zoom level in current implementation

package kripos.math.circle;

import java.awt.geom.Point2D;
import java.io.ByteArrayOutputStream;
import java.io.PrintStream;
import java.util.LinkedList;
import java.util.List;
import java.util.Locale;

/**
 * Representation of a circle, with methods to calculate numbers relative to another circle.
 *
 * @author oysteine
 */

class Circle {

  /**
   * Center of circle;
   */
  public final Point2D origo;

  /**
   * Radius of circle
   */
  public double rad;

  protected int intersectionCount = 0;

  /**
   * @param x cartesian x coord
   * @param y cartesian y coord
   * @param rad length of radius
   */
  public Circle(double x, double y, double rad) {
    this.origo = new Point2D.Double(x, y);
    this.rad = rad;
  }

  /**
   *
* @param angle radians from x axis
* @return point on circle circumference
*/
public Point2D getPoint(double angle) {
    double x = origo.getX() + rad * Math.cos(angle);
    double y = origo.getY() + rad * Math.sin(angle);
    return new Point2D.Double(x, y);
}

/**
 * @param inter target: a intersection on this circle
 * @return the angle between the x axis and the line from
 *         this circle's origo point to <code>inter</code>'s
 *         point
 */
double getAngle(Intersection inter) {
    assert this == inter.c1 || this == inter.c2;
    double opposite = inter.point.getY() - origo.getY(); //
    opposite
double hyp = rad;
    if (inter.isAbove(origo)) {
        if (inter.isRightOf(origo)) {
            return Math.asin(opposite / hyp);
        } else {
            return Math.PI - Math.asin(opposite / hyp);
        }
    } else {
        if (inter.isRightOf(origo)) {
            return 2 * Math.PI + Math.asin(opposite / hyp);
        } else {
            return Math.PI - Math.asin(opposite / hyp);
        }
    }
}

/**
 * @param inter
 * @return <code>true</code> iff <code>inter</code> is
 *         neither inside
 *         this circle's area nor on its circumference
 */
boolean isOutside(Intersection inter) {
    return !inter.belongsTo(this) &&
    origo.distance(inter.point) > rad;
}

}
* @return `<code>true</code>` iff `<code>c</code>` is completely within this circle’s area or on its circumference

```java
boolean isWithin(Circle other) {
    if (other == this) {
        return true;
    }

    double dist = origo.distance(other.origo);
    return dist + other.rad <= rad;
}
```

```java
/**
 * @param other not `<code>null</code>`
 * @return intersection points, or `<code>null</code>` if and only if circles do not intersect
 * at `<i>`two`/i` points
 */
List<Intersection> getIntersections(Circle other) {
    assert other != null;

    double maxDist = rad + other.rad;
    double dist = origo.distance(other.origo);
    if (dist >= maxDist) {
        return null;
    }
    if (dist + Math.min(rad,other.rad) < Math.max(rad,other.rad)) {
        return null;
    }

    double x1 = origo.getX();
    double y1 = origo.getY();
    double r1 = rad;
    double x2 = other.origo.getX();
    double y2 = other.origo.getY();
    double r2 = other.rad;

    double d = Math.sqrt(Math.pow(x2-x1,2) + Math.pow(y2-y1,2));
    double ixPart1 = (x2+x1) / 2 + (x2-x1) * (r1*r1-r2*r2) / (2*d*d);
    double ixPart2 = ( (y2-y1) / (2*d*d) ) *
        Math.sqrt((Math.pow(r1+r2,2)-d*d) * (d*d-Math.pow(r2-r1,2)) );
    double iyPart1 = (y2+y1) / 2 + (y2-y1) * (r1*r1-r2*r2) / (2*d*d);
    double iyPart2 = ( (x2-x1) / (2*d*d) ) *
        Math.sqrt((Math.pow(r1+r2,2)-d*d) * (d*d-Math.pow(r2-r1,2)) );
```
Point2D first = new Point2D.Double(ixPart1+ixPart2, iyPart1-iyPart2);
Point2D second = new Point2D.Double(ixPart1-ixPart2, iyPart1+iyPart2);

List<Intersection> inters = new LinkedList<Intersection>();
inters.add(new Intersection(first, this, other));
inters.add(new Intersection(second, this, other));
return inters;

/**
 * @param x
 * @param y
 * @return distance from the given point to circumference
 */
public double getMargin(double x, double y) {
    double margin =
        Math.pow(x - origo.getX(), 2) +
        Math.pow(y - origo.getY(), 2) -
        Math.pow(rad, 2);
    return margin;
}

@Override
public String toString() {
    ByteArrayOutputStream tmp = new ByteArrayOutputStream();
    PrintStream stream = new PrintStream(tmp);
    stream.printf(Locale.US, "Circle(%.2f, %.2f, %.2f), ",
        origo.getX(), origo.getY(), rad);
    return tmp.toString();
}

//class
C.6.12 Intersection

//legacy code. only works in 2D. originally meant used with a flat map based on UTM coordinates
//turned out to be not accurate enough when spanning multiple UTM zones

//only used for setting zoom level in current implementation

package kripos.math.circle;

import java.awt.geom.Point2D;
import java.io.ByteArrayOutputStream;
import java.io.PrintStream;
import java.util.Locale;

/**
 * Representing one intersection between two circles.
 *
 * @author oysteine
 */
public class Intersection {
    /** Where the two circles cross. */
    public final Point2D point;
    final Circle c1;
    final Circle c2;

    /**
     * @param point the intersection
     * @param circle1 one of the participating circles
     * @param circle2 the other participating circle
     */
    public Intersection(Point2D point, Circle circle1, Circle circle2) {
        this.point = point;
        this.c1 = circle1;
        this.c2 = circle2;
    }

    boolean isLeftOf(Intersection c) {
        return isLeftOf(c.point);
    }
    boolean isLeftOf(Point2D p) {
        return point.getX() < p.getX();
    }
    boolean isRightOf(Intersection c) {
        return isRightOf(c.point);
    }
    boolean isRightOf(Point2D p) {
        return point.getX() > p.getX();
    }
```java
50 }
51 
52 boolean isAbove(Intersection c) {
53     return isAbove(c.point);
54 }
55 
56 boolean isAbove(Point2D p) {
57     return point.getY() > p.getY();
58 }
59 
60 boolean isBelow(Intersection c) {
61     return isBelow(c.point);
62 }
63 
64 boolean isBelow(Point2D p) {
65     return point.getY() < p.getY();
66 }
67 
68 /**
69 * @param c target: an intersection above this
70 * @return the angle between the x axis and the line from
71 *     this intersection’s point to <code>c</code>’s
72 * point
73 */
74 double getAngle(Intersection c) {
75     assert !isAbove(c);
76     double opposite = c.point.getY() - point.getY(); //
77     opposite
78     double hyp = point.distance(c.point);
79     if (c.isRightOf(this)) {
80         return Math.asin(opposite / hyp);
81     } else {
82         return Math.PI - Math.asin(opposite / hyp);
83     }
84 }
85 
86 /**
87 * @param c
88 * @return <code>true</code> iff <code>c</code> is one of the
89 * circles in this intersection
90 */
91 public boolean belongsTo(Circle c) {
92     return c == c1 || c == c2;
93 }
94 
95 /**
96 * @param c
97 * @return the circle that is not <code>c</code>
98 */
99 public Circle getOtherCircle(Circle c) {
100     assert c == c1 || c == c2;
101     return c == c1 ? c2 : c1;
102 }
103 ```
@Override
public String toString() {
    ByteArrayOutputStream tmp = new ByteArrayOutputStream();
    PrintStream stream = new PrintStream(tmp);
    stream.printf(Locale.US, "p(x=%.2f, y=%.2f), c1=%s, c2=%s",
                  point.getX(), point.getY(), c1.toString(), c2.toString());
    return tmp.toString();
}
} // class
C.6.13 Intersector

// legacy code. only works in 2D, originally meant used with a flat map based on UTM coordinates

// turned out to be not accurate enough when spanning multiple UTM zones

// only used for setting zoom level in current implementation

package kripos.math.circle;

import java.awt.geom.Point2D;
import java.util.Arrays;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;

/**
 * Calculate intersections from many circles.
 * @author oysteine
 *
 */
public class Intersector {

    /**
     * @param inters
     * @return length of circumference of convex hull created by <code>inters</code>
     */
    static public double getPolygonLength(List<Intersection> inters) {
        double length = 0.0;

        if (inters.size() == 0) {
            // circumference of single circle
            Intersection i = inters.get(0);
            assert i.c1 == i.c2;
            return 0.0;
        }
        // return 2.0 * Math.PI * i.c1.rad;

        List<Intersection> tmp = new LinkedList<Intersection>(inters);
        Intersection prev = tmp.remove(0);
        tmp.add(prev);
        for (Intersection next : tmp) {
            length += prev.point.distance(next.point);
            prev = next;
        }

        return length;
    }
}
static public double getPolygonArea(List<Intersection> inters) {
    if (inters.size() == 0) {
        return 0.0;
    }
    double area = 0.0;
    List<Intersection> tmp = new LinkedList<Intersection>(inters);
    Intersection prev = tmp.remove(0);
    tmp.add(prev);
    for (Intersection next : tmp) {
        double x1 = prev.point.getX();
        double y1 = prev.point.getY();
        double x2 = next.point.getX();
        double y2 = next.point.getY();
        area += x1*y2 - x2*y1;
        prev = next;
    }
    area *= 0.5;
    return area;
}

static public Point2D getCentroid(List<Intersection> inters) {
    if (inters.size() < 2) {
        return null;
    }
    double area = getPolygonArea(inters);
    double x = 0.0;
    double y = 0.0;
    List<Intersection> tmp = new LinkedList<Intersection>(inters);
    Intersection prev = tmp.remove(0);
    tmp.add(prev);
    for (Intersection next : tmp) {
        double x1 = prev.point.getX();
        double y1 = prev.point.getY();
        double x2 = next.point.getX();
        double y2 = next.point.getY();
        area += x1*y2 - x2*y1;
        prev = next;
    }
    area *= 0.5;
    return area;
}
double x2 = next.point.getX();
double y2 = next.point.getY();

double determinant = x1*y2 - x2*y1;
x += (x1+x2) * determinant;
y += (y1+y2) * determinant;
prev = next;
} else {
x /= (6 * area);
y /= (6 * area);

return new Point2D.Double(x,y);
}

/**
 * @param circles
 * @return exact size of area covered by all circles
 */
static public double getExactArea(List<Circle> circles) {
    if (circles.size() == 1) {
        Circle c = circles.get(0);
        return Math.PI * c.rad * c.rad;
    }
    List<Intersection> inters = getIntersections(circles);
    if (inters.size() == 0) {
        return 0.0;
    }

double area = getPolygonArea(inters);

List<Intersection> tmp = new LinkedList<Intersection>(inters);
Intersection prev = tmp.remove(0);
tmp.add(prev);
for (Intersection next : tmp) {
    ArcInfo arc = getArc(prev, next);
    double asize = arc.a2 - arc.a1;
    double sliceArea = asize / 2 * Math.pow(arc.c.rad, 2);
    double triangleArea = Math.pow(arc.c.rad, 2) *
        Math.cos(asize/2) * Math.sin(asize/2);
    area += sliceArea - triangleArea;
    prev = next;
}

return area;
}
```java
/**
 * @param inters
 * @param distance max distance between each point
 * @return all original <code>inters</code> point plus points
 * located on the circumferences on the circles,
 * with maximum <code>distance</code> length between
 * two adjacent points
 */

public static List<Intersection> getMorePoints(
    List<Intersection> inters, double distance) {
    List<Intersection> populated = new LinkedList<Intersection>();
    if (inters.size() == 0) {
        return populated;
    }

    //System.out.println("GETTING MORE POINTS!");
    List<Intersection> tmp = new LinkedList<Intersection>(inters);
    Intersection prev = tmp.remove(0);
    tmp.add(prev);
    for (Intersection next : tmp) {
        populated.add(prev);
        ArcInfo arc = getArc(prev, next);
        //System.out.printf(arc.c + ": a1=%.2f, a2=%.2f\n", arc.a1, arc.a2);
        double asize = arc.a2 - arc.a1;
        double arclen = arc.c.rad * asize;
        double adelta = asize / (arclen / distance);
        for (double anew = arc.a1 + adelta; anew < arc.a2; anew += adelta) {
            //System.out.printf(" adding %.2f\n", anew);
            Point2D p = arc.c.getPoint(anew);
            populated.add(new Intersection(p, arc.c, arc.c));
        }
    }
    prev = next;
}

return populated;

private static class ArcInfo {
    private final Circle c;
    private final double a1;
    private final double a2;
}
```
private ArcInfo(Circle c, double a1, double a2) {
  this.c = c;
  this.a1 = a1;
  this.a2 = a2;
}

private static ArcInfo getArc(Intersection prev, Intersection next) {
  // find potential circles
  Circle c = prev.c1;
  Circle cc = null;
  if (!next.belongsTo(c)) {
    c = prev.c2;
  } else {
    cc = prev.c2;
    if (!next.belongsTo(cc)) {
      cc = null;
    }
  }

  // find degrees and wanted circle
  double a1 = c.getAngle(prev);
  double a2 = c.getAngle(next);
  if (a2 <= a1) {  // equals to support a single circle with a
    a2 += 2 * Math.PI;
  }

  if (cc != null && cc != c &&
      (a2 - a1 > Math.PI || c.rad < cc.rad)) {
    double b1 = cc.getAngle(prev);
    double b2 = cc.getAngle(next);
    if (b2 < b1) {
      b2 += 2 * Math.PI;
    }
    if (b2 - b1 < Math.PI ||
        cc.rad < c.rad) {
      c = cc;
      a1 = b1;
      a2 = b2;
    }
  }

  return new ArcInfo(c, a1, a2);
}
* @param circles
* @return all points where circles cross to make up
* the area all circles overlap
*/

public static List<Intersection> getIntersections(List<Circle> circles) {
    List<Intersection> inters = new LinkedList<Intersection>();
    if (circles.size() == 0) {
        return inters;
    }

    // set intersections at each circle
    List<Circle> sources = new LinkedList<Circle>(circles);
    List<Circle> targets = new LinkedList<Circle>();
    Circle firstSource = sources.remove(0);
    targets.add(firstSource);

    // add dummy intersection
    inters.add(new Intersection(firstSource.getPoint(0), firstSource, firstSource));
    firstSource.intersectionCount += 2;

    while (!sources.isEmpty()) {
        Circle source = sources.remove(0);

        // remove previous intersections that fall outside new source
        Iterator<Intersection> iterator = inters.iterator();
        while (iterator.hasNext()) {
            Intersection inter = iterator.next();
            if (source.isOutside(inter)) {
                iterator.remove();
                inter.c1.intersectionCount --;
                inter.c2.intersectionCount --;
            }
        }

        // add new intersections from source
        for (Circle target : targets) {
            List<Intersection> newInters = target.getIntersections(source);
            if (newInters != null) {
                for (Intersection newInter : newInters) {
                    boolean valid = true;
                    for (Circle target2 : targets) {
                        if (target2.isOutside(newInter)) {
                            valid = false;
                            break;
                        }
                    }
                    if (valid) {
inters.add(newInter);
newInter.c1.intersectionCount++;  
newInter.c2.intersectionCount++;  
}
}
}
}
if (inters.size() == 0) {
  // add dummy intersection if source is within all targets
  boolean withinAll = true;
  for (Circle target : targets) {
    if (!target.isWithin(source)) {
      withinAll = false;
      break;
    }
  }
  // add dummy intersection
  if (withinAll) {
    inters.add(new Intersection(
      source.getPoint(0), source, source));
    source.intersectionCount += 2;
  }
} else if (inters.size() > 1) {
  // remove dummy intersection
  Intersection test = inters.get(0);
  if (test.c1 == test.c2) {
    inters.remove(0);
    test.c1.intersectionCount -= 2;
  }
} // add source as target
targets.add(source);
// remove targets without any intersections
Iterator<Circle> citer = targets.iterator();
while (citer.hasNext()) {
  Circle target = citer.next();
  assert target.intersectionCount >= 0 :
  target.toString() + "intersCount=" + target.intersectionCount;
  if (target.intersectionCount == 0) {
    citer.remove();
  }
}
if (inters.size() == 0) {
  return inters;
}
// find lowest point to start convex polygon creation
Intersection bottom = inters.get(0);
for (Intersection inter : inters) {
    if (inter.isBelow(bottom)) {
        bottom = inter;
    }
}
inters.remove(bottom);

// find angles from bottom to rest of intersections
AngledIntersection[] angles = new AngledIntersection[inters.size()];
int i = 0;
for (Intersection inter : inters) {
    angles[i++] = new AngledIntersection(
        inter,
        bottom.getAngle(inter));
}

// sort angles and add
List<Intersection> sorted = new LinkedList<Intersection>();
sorted.add(bottom);
Arrays.sort(angles);
for (AngledIntersection tmp : angles) {
    sorted.add(tmp.inter);
}
return sorted;

private static class AngledIntersection implements Comparable<AngledIntersection> {
    private final Intersection inter;
    private final double angle;
    public AngledIntersection(Intersection inter, double angle) {
        this.inter = inter;
        this.angle = angle;
    }

    public int compareTo(AngledIntersection other) {
        if (angle < other.angle) {
            return -1;
        } else if (angle > other.angle) {
            return 1;
        } else {
            // TODO: compare distance
            return 0;
        }
    }
}
C.7 DB Classes

C.7.1 DBCreator

```java
package kripos.tools;

import java.net.InetAddress;
import java.net.UnknownHostException;
import java.sql.Connection;
import java.sql.DatabaseMetaData;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import kripos.geo.Landmark;

/**
 * Creates and fills the database used by agents for storing geolocation information
 * Any existing database and content will be dropped!
 *
 * @author oysteine
 * @version
 *
 */
public class DBCreator {
    private String dbmsPath; // TODO not used yet
    private String user; // TODO not used yet
    private String pw; // TODO not used yet
    private String myName;

    /**
     * Creates the database tables relative to the landmark host
     *
     */
    public DBCreator() {
        InetAddress iadr;
        try {
            iadr = InetAddress.getLocalHost();
            myName = iadr.getCanonicalHostName();
        } catch (UnknownHostException e) {
            System.out.println("Unable to get local fqdn hostname");
            e.printStackTrace();
        }
    }

    /**
     * Establishes a connection to the database
     *
     * @return con connection to the database
     */
```
private Connection connect() throws SQLException, ClassNotFoundException {
    Connection c = null;
    Class.forName("org.hsqldb.jdbcDriver");
    c = DriverManager.getConnection("jdbc:hsqldb:hsql://localhost/xdb", "sa", "");
    return c;
}

/**
 * Creates and fills the table containing information about landmarks
 *
 * @param Database connection con
 */
private void createLandmarkTable(Connection con) throws SQLException {
    String landmarkTable = "CREATE TABLE LANDMARKS " +
    "(NAME VARCHAR(32) NOT NULL, IPADR VARCHAR(39) NOT NULL,
     CHECKED TIMESTAMP, " +
    "DISTANCE_KM DOUBLE NOT NULL, LATITUDE DOUBLE NOT NULL,
     LONGITUDE DOUBLE NOT NULL, MIN_RTT DOUBLE, " +
    "AVG_RTT DOUBLE, C1 DOUBLE, EPSILON DOUBLE, HASH VARCHAR(64),
     PRIMARY KEY (NAME, IPADR))";

    Statement stmt = con.createStatement();
    stmt.executeUpdate(landmarkTable);
    //fill table
    LandmarkReader lr = new LandmarkReader();
    ArrayList<Landmark> landmarks = lr.distance(myName);
    for (int i=0;i<landmarks.size();i++){
        Landmark l = landmarks.get(i);
        double distance = l.getDistance();
        double latitude = l.getGeoPosition().getLatitude();
        double longitude = l.getGeoPosition().getLongitude();
        String IP = l.getIP();
        String name = l.getName();
        String landmarkAdd = "INSERT INTO LANDMARKS VALUES (" +
                     "'"+name+"','"+IP+"','null,"+distance+","+latitude+
                     ","+longitude+"','" +
                     "999999999"+
                     "," +-1+"','"+-1+"','"+-1+","'A')";

        stmt.executeUpdate(landmarkAdd);
    }
}

/**
 * Creates the table containing information about traced hosts
 *
 * @param Database connection con
 */
/*
private void createTraceTable(Connection con) throws SQLException{
    // contains traced hosts and info
    String traceTable = "CREATE TABLE TRACED (NAME VARCHAR(32), IPADR VARCHAR(39), CHECKED TIMESTAMP NOT NULL, MIN_RTT DOUBLE NOT NULL, AVG_RTT DOUBLE, C1 DOUBLE, EPSILON DOUBLE, HASH VARCHAR(64), PRIMARY KEY(IPADR))";
    Statement stmt;
    stmt = con.createStatement();
    stmt.executeUpdate(traceTable);
}
/**
* Creates the table containing misc information,
* @param Database connection con
*/
private void createMiscTable(Connection con) throws SQLException{
    // contains misc data, when last bestline etc.
    String miscTable = "CREATE TABLE MISC (NAME VARCHAR(32), IPADR VARCHAR(39), LAST_BESTLINE TIMESTAMP, BESTLINE_M DOUBLE, BESTLINE_B DOUBLE, LATITUDE DOUBLE, LONGITUDE DOUBLE)";
    Statement stmt = con.createStatement();
    stmt.executeUpdate(miscTable);
    // create table
    LandmarkReader lr = new LandmarkReader();
    Landmark l = lr.getSingleLandmark(myName);
    double latitude = l.getGeoPosition().getLatitude();
    double longitude = l.getGeoPosition().getLongitude();
    String IP = l.getIP();
    String miscAdd = "INSERT INTO MISC VALUES ('"+myName+"','"+IP+"',null,0,0,"+latitude+"","+longitude+")";
    stmt.executeUpdate(miscAdd);
}
/**
* @param args
* @throws SQLException
*/
public static void main(String[] args) {
    DBCreator dbCreate = new DBCreator();
    try{
        Connection con = dbCreate.connect();
        Statement stmt;
    }
stmt = con.createStatement();
DatabaseMetaData dbmd = con.getMetaData();
ResultSet rs1 = dbmd.getTables(null, null, null, null);
// drop all existing normal tables
while(rs1.next()){
    String tableName = rs1.getString("TABLE_NAME");
    String tableType = rs1.getString("TABLE_TYPE");
    if(tableType.equalsIgnoreCase("TABLE")){
        stmt.execute("DROP TABLE " + tableName);
    }
}
// create and fill tables
dbCreate.createLandmarkTable(con);
dbCreate.createTraceTable(con);
dbCreate.createMiscTable(con);
con.close();
System.out.println("Database created successfully!");
System.exit(0);
}
} catch (SQLException se) {
    System.out.println("Database creation FAILED!");
    se.printStackTrace();
    System.exit(1);
} catch (ClassNotFoundException ce) {
    System.out.println("Database creation FAILED!");
    ce.printStackTrace();
    System.exit(1);
}
} // class
package kripos.tools;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;

/**
 * @author oysteine
 *
 */
public class DBStop {

    public DBStop() {
    }

    public static void main(String[] args) {
        DBStop dbs = new DBStop();
        try {
            Connection con = dbs.connect();
            Statement st = con.createStatement();
            st.execute("SHUTDOWN");
        } catch (SQLException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        DBStop dbs = new DBStop();
        try {
            Connection con = dbs.connect();
            Statement st = con.createStatement();
            st.execute("SHUTDOWN");
        } catch (SQLException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }

    private Connection connect() throws SQLException, ClassNotFoundException {
        Class.forName("org.hsqldb.jdbcDriver");
        Connections c = new DriverManager.getConnection("jdbc:hsqldb:hsq1://localhost/xdb", "sa", ""); //FIXME
        return c;
    }

    }
C.7.3  LandmarkReader

```java
package kripos.tools;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.net.InetAddress;
import java.util.ArrayList;
import kripos.geo.Landmark;
import com.bbn.openmap.LatLonPoint;
import com.bbn.openmap.proj.Length;
import com.bbn.openmap.proj.coords.UTMPoint;

/**< *
* Tool to read landmark information from file.
* Includes method to calculate distance between landmarks
* @author oysteine
* @version 1.1
*/
public class LandmarkReader {
    private ArrayList<Landmark> landmarks = new ArrayList<Landmark>();

    /**
     * Creates a LandmarkReader and reads in landmark information
     * from file
     */
    public LandmarkReader() {
        convert();
    }

    /**
     * Calculates and returns the distances to all landmarks
     * from the landmark with the name provided.
     * @param name of landmark to compute distances from
     * @return list of landmarks with distances.
     */
    public ArrayList<Landmark> distance(String landmarkName) {
        Landmark from = null;
        for (int i = 0; i < landmarks.size(); i++) {
            if (landmarkName.equalsIgnoreCase(landmarks.get(i).getName())) {
                from = landmarks.remove(i);
                break;
            }
        }
        for (int i = 0; i < landmarks.size(); i++) {
```
double radDistance = from.getGeoPosition().distance(
    landmarks.get(i).getGeoPosition());

Length converter = Length.KM;
landmarks.get(i).setDistance(converter.fromRadians(
    radDistance));
}
return landmarks;
}
/**
 * Get information about a single landmark in the form of
 * Landmark object
 *
 * @param landmarkName the name of the
 * @return the Landmark object for the landmark queried for
 */
public Landmark getSingleLandmark(String landmarkName){
    Landmark result = null;
    for(int i=0;i<landmarks.size();i++){
        if(landmarkName.equalsIgnoreCase(landmarks.get(i).getName())){
            result = landmarks.remove(i);
            break;
        }
    }
    return result;
}
/**
 * Reads landmarkinformation from file
 * | is used as field separator
 * lines starting with # are ignored
 */
private void convert(){
    try {
        FileReader fr = new FileReader("maalepaaler.txt");
        BufferedReader br = new BufferedReader(fr);
        String temp = null;
        while((temp = br.readLine()) !=null){
            if('#' == temp.charAt(0)){ //ignore lines starting with #
                }
            else{
                int firstCutPoint = temp.indexOf(' |');
                String name = temp.substring(0, firstCutPoint);
                int secondCutPoint = temp.indexOf(' |', firstCutPoint +1);
                int UTMZone = Integer.parseInt(temp.substring( firstCutPoint+1, secondCutPoint));
                int thirdCutPoint = temp.indexOf(' |', secondCutPoint +1);
                int UTMNorthing = Integer.parseInt(temp.substring( secondCutPoint+1, thirdCutPoint));
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
int UTMEasting = Integer.parseInt(temp.substring(thirdCutPoint+1));
UTMPoint utmp = new UTMPoint(UTMNorthing, UTMEasting, UTMZone,'N');
LatLonPoint llp = utmp.toLatLonPoint();

//get the current IP-address of the hostname
InetAddress iadr = InetAddress.getByName(name);
String IP = iadr.getHostAddress();

Landmark l = new Landmark(name, llp, IP);
landmarks.add(l);
C.7.4 Landmark

package kripos.geo;

import java.util.Date;

import com.bbn.openmap.LatLonPoint;

/**
 * Contains information about a single Landmark, 
 * relative to the position of the owner of this instance.
 * 
 * @author oysteine
 * @version 1.0
 */
public class Landmark {
  private String myName;
  private LatLonPoint geoPosition; //TODO
  private double myDistance;
  private String myIP;
  private Date lastChecked;
  private double minRTT = 0;
  private double avgRTT;
  private double C1;
  private double epsilon;
  private String hashedTimestamp; //TODO
  
  /**
   * 
   */
  public Landmark(String IP) {
    myIP = IP;
  }
  
  /**
   * 
   */
  public Landmark(String name, LatLonPoint llp, String IP) {
    myName = name;
    myIP = IP;
    geoPosition = llp;
  }
  
  /**
   * 
   */
  public Landmark(String name, double distance, String IP) {
    myName = name;
    myDistance = distance;
    myIP = IP;
  }
  
  /**
   * @return the geoPosition
   */
}
/**
 * @return the GeoPosition
 */
public LatLonPoint getGeoPosition() {
    return geoPosition;
}

/**
 * @param myDistance the myDistance to set
 */
public void setDistance(double myDistance) {
    this.myDistance = myDistance;
}

/**
 * @return the avgRTT
 */
public double getAvgRTT() {
    return avgRTT;
}

/**
 * @param avgRTT the avgRTT to set
 */
public void setAvgRTT(double avgRTT) {
    this.avgRTT = avgRTT;
}

/**
 * @return the c1
 */
public double getC1() {
    return C1;
}

/**
 * @param c1 the c1 to set
 */
public void setC1(double c1) {
    C1 = c1;
}

/**
 * @return the epsilon
 */
public double getEpsilon() {
    return epsilon;
}

/**
 * @param epsilon the epsilon to set
 */
public void setEpsilon(double epsilon) {
    this.epsilon = epsilon;
}
public String getHashedTimestamp() {
    return hashedTimestamp;
}

public void setHashedTimestamp(String hashedTimestamp) {
    this.hashedTimestamp = hashedTimestamp;
}

public Date getLastChecked() {
    return lastChecked;
}

public void setLastChecked(Date lastChecked) {
    this.lastChecked = lastChecked;
}

public double getMinRTT() {
    return minRTT;
}

public void setMinRTT(double minRTT) {
    this.minRTT = minRTT;
}

public double getDistance() {
    return myDistance;
}

public void SetDistance(double distance) {
    myDistance = distance;
}
```java
public String getIP() {
    return myIP;
}

public String getName() {
    return myName;
}
```

//class
C.8 Scripts used for Managing the System

C.8.1 Unidist

```bash
#!/bin/bash
hostfile="/diplom/shell/unihosts"
uniuser="oysteine"
content="/diplom/shell/disttest"
files=`find $content -maxdepth 1 -type f`
directories=`find $content -mindepth 1 -maxdepth 1 -type d`
destination="/"
if [[ ! -e "$hostfile" ]]
then
    printf "${hostfile##*/}non-existent"
    exit 1
fi
for host in $(cat $hostfile)
do
    printf "Copying to host $host *************************
    for directoryLine in $directories
do
        scp -r "$directoryLine/" $uniuser@$host:$destination
done
    for fileLine in $files
do
        scp "$fileLine" $uniuser@$host:$destination
done
    printf "Copied current version to all unihosts"
    printf 

```

C.8.2 Unirun

```bash
#!/bin/bash
hostfile="/diplom/shell/unihosts"
uniuser="oysteine"
classpath1="/home/oysteine/hsqldb/lib/hsqldb.jar:/home/oysteine/
jade/lib/jade.jar:/home/oysteine/jade/lib/jadeTools.jar:/
home/oysteine/jade/lib/iiop.jar:/home/oysteine/jade/lib/
commoms-codec-1.3.jar"
classpath2="/home/oysteine/jade/lib/jade.jar:.:/home/oysteine/
jade/lib/jadeTools.jar:/home/oysteine/jade/lib/iiop.jar:/
home/oysteine/jade/lib/commons-codec-1.3.jar:/home/oysteine/
openmap/lib/openmap.jar:/home/oysteine/Jama-1.0.2.jar:/home/
oysteine/hsqldb/lib/hsqldb.jar"
sjef="futurum01.item.ntnu.no"
```
C.8.3 Unistop

```bash
#!/bin/bash
hostfile="/diplom/shell/unihosts
uniuser="oysteine"
classpath1="~/hsqldb/lib/hsqldb.jar:~/geolocate.jar"

if [[ ! -e "$hostfile" ]]
  then
    printf "${hostfile##*/}non-existent"
  exit 1
fi

for host in $(cat $hostfile)
do
  ssh $uniuser@$host java -cp $classpath1: kripos.tools.DBStop
  ssh $uniuser@$host killall java
done

printf "system stopped"
```

C.8.4 Unikill

```bash
#!/bin/bash
hostfile="/diplom/shell/unihosts
uniuser="oysteine"
classpath1="~/hsqldb/lib/hsqldb.jar:~/geolocate.jar"

if [[ ! -e "$hostfile" ]]
```
then
printf "${hostfile##*/}non-existent"
exit 1
fi
for host in $(cat $hostfile)
do
  ssh $uniuser@$host java -cp $classpath1: kripos.tools.DBSstop
  ssh $uniuser@$host killall -9 java
done

C.8.5 Unicdb

#!/bin/bash
hostfile="~/diplom/shell/unihosts
uniuser="oysteine"
classpath1=":.:/home/oysteine/hsqldb/lib/hsqldb.jar:/home/oysteine/openmap/lib/openmap.jar"
if [[ ! -e "$hostfile" ]]
then
  printf "${hostfile##*/}non-existent"
  exit 1
fi
for host in $(cat $hostfile)
do
  printf "$host***************"
  printf "\n"
  ssh $uniuser@$host java -cp $classpath1: org.hsqldb.Server -
  database.0 mydb -dbname.0 xdb &
  sleep 2
  ssh $uniuser@$host java -cp $classpath1 kripos.tools.DBCreator &
done

C.9 JADE properties files

C.9.1 JADE-S main.conf

# ---- JADE configuration ----

# ------ Services ------
services=
  jade.core.security.SecurityService;
  jade.core.security.signature.SignatureService;
  jade.core.security.encryption.EncryptionService;
  jade.core.event.NotificationService

267
# ------ Agents ------
agents=kripos-rma:jade.tools.rma.rma

# ------ Security configuration ------
# ---- Permission ----
# Permission Policy file
java.security.policy=policy.txt

# ---- Authentication ----
# - Type of Prompt
jade.security.authentication.logincallback=Cmdline
# - if Cmdline, use this user/pass -
# owner=kripos:password
# - Auth module
jade.security.authentication.loginmodule=Simple
# - if Simple, use this password file
jade.security.authentication.loginsimplecredfile=passwords.txt

# - JAAS configuration file -
java.security.auth.login.config=jaas.conf

# ---- end JADE configuration ----
C.9.2 jaas.conf

```java
/*
 * JAAS configuration file
 */

Simple {
  jade.core.security.authentication.SimpleLoginModule required;
};

Unix {
  com.sun.security.auth.module.UnixLoginModule required;
};

NT {
  com.sun.security.auth.module.NTLoginModule required;
};

Kerberos {
  com.sun.security.auth.module.Krb5LoginModule required;
};
```

C.9.3 policy.txt

```java
grant codebase "file:/home/oysteine/jade/add-ons/security/lib/
  jadeSecurity.jar" {
  permission java.security.AllPermission; }
grant codebase "file:/home/oysteine/jade/lib/jade.jar" {
  permission java.security.AllPermission; }
grant codebase "file:/home/oysteine/jade/lib/jadeTools.jar" {
  permission java.security.AllPermission; }

// --- Policy on the MAIN container ---

grant principal jade.security.Name "kripos" {
  permission java.security.AllPermission;
};
```

C.9.4 passwords.txt

```java
kripos test
```
Appendix D

Map Projections and Reference Systems

In this Appendix some of the geographical properties of the earth with regard to GIS and maps are described. A basic understanding of these properties and different ways to model the earth and locations on it is necessary to appreciate some of the discussion with regard to the choice of geographical toolkit, and also the distance calculations involved in converting delay measurements to geographical distance.

Most of the effects on geolocation due to the differences among the reference systems and earth models described below are minor. Current geolocation techniques are not accurate enough for these effects to be important. They are only included here for completeness and future reference.

D.1 Map Projections

All maps of the earth are based on map projections. A map projection is any method used in cartography to represent the two-dimensional curved surface of the earth or other body on a plane [map06]. A surface that can be unfolded into a flat plane without any form of distortion is called a developable surface. Unfortunately the earth is an approximately elliptical spheroid, a form that is not a developable surface. Any projection used to "flatten" it will incur some distortions. The type and severity of the distortions depends on the projection used. Different projections are designed to preserve certain properties, as it is impossible to avoid distortion all together.

For map projections, and particularly for GIS systems, different approximations of the shape of the earth are used, with different distortion properties. This, in
addition to the choice of map projection, leads to slightly different coordinates being assigned to the same location, depending on the earth model and map projection used. These differences influence the accuracy of distance calculation in our system.

D.2 Geographical Reference Systems

Not only are there different map projections based on different earth models. Several different reference systems have also been developed. Most of these reference systems define what model of the earth is to be used to avoid ambiguity between locations and their coordinates. However, conversion between the different systems may introduce inaccuracies.

D.2.1 World Geodetic System (WGS)

The World Geodetic System defines a fixed global reference frame for the earth. It was originally conceived in 1960 and named WGS60. The latest revision is WGS 84 dating from 1984, although with several minor updates, the last from 2004. WGS84 is used by the Global Positioning System (GPS). It is geocentric and globally consistent within ±1 m. The longitude positions on WGS84 differ somewhat from older datums, the zero meridian of WGS84 is about 100 meters to the east of the traditional zero meridian at Greenwich [wgs06].

D.2.2 Universal Transverse Mercator (UTM)

UTM is a grid-based method of specifying locations on the surface of the Earth. It differs from the method of latitude and longitude in several respects. Unlike for latitude and longitude, there is no physical frame of reference for the UTM grid. Latitude is determined by the earth’s polar axis and longitude is determined by its rotation. UTM coordinates are simply defined by the grid used [Dut06].

The UTM system is not a map projection, it is based on a collection of sixty longitude zones, where each zone is based on a specifically defined Transverse Mercator projection. The WGS84 ellipsoid is used as the underlying earth model. UTM does not cover the entire surface of the earth, the zones do not cover the areas north of 84° and south of 80°. Each of the 60 zones is 6° longitude wide and centered over a meridian of longitude. Zone 1 is defined as longitude 180° to 174° W. Zone numbers increase in an easterly direction. Each zone maps a region of large north-south extent with a low amount of distortion, below 1:1,000 inside each zone, distortion is higher at the edges of a zone. The longitude zones are partitioned into 20 latitude zones, each 8 degrees high [utm06, Dea06].
The partition into longitude and latitude zones is globally uniform, except in two areas; on the southwest coast of Norway, the zone 32V is extended westward, and the zone 31V is correspondingly shrunk to cover only open water, see Figure D.1. Also, in the region around Svalbard, the longitude zones are given double their normal width. This has implications for the accuracy of the locations of some of our measurement nodes, since their locations were provided in UTM format by Uninett. The UTM system’s accuracy is rated as 1:2,500 [Dea06]. This means that the true length of a distance measured to be 2,500km lies between 2,499km and 2,501km. The accuracy will of course be lower when the zone size is doubled. Conversion between longitude/latitude and UTM involves rather complex equations, and different implementations may take shortcuts leading to small inaccuracies.

![Figure D.1: The extended UTM zone 32V and the shrunk 31V.](image)

### D.3 Great-circle Distance

Great-circle distance is the shortest distance between any two points on the surface of a sphere. It is defined by a great circle with the same center as the sphere: Between any two points on a sphere, there is a unique great circle, except if the two points are exactly opposite each other, in which case there is an infinite set of matching great circles. The two points separate the great circle into two arcs. The length of the shorter arc is the great-circle distance between the points.

Great-circle distances can be used to calculate the distance between locations on earth, if the form of the earth is approximated as a sphere. Using a sphere with a radius of 6372.795 km this approximation results in an error of up to about 0.5% [gre06].