NTNU
Innovation and Creativity

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).

Assignment given: 28. June 2006
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

List of Figures

List of Tables

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 geolocation 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 web-sites. 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 principle 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.

```
Information related to '129.241.0.0 - 129.241.255.255'

inetnum:        129.241.0.0 - 129.241.255.255
netname:        NTNU
descr:          Hogskoleringen 1
descr:          NO-7491 Trondheim
country:        NO

irt:            IRT-UNINETT-CERT
address:        UNINETT CERT
address:        Abels gate 5
address:        N7465
address:        Trondheim
address:        Norway
source:         RIPE
```

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. Gueye 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 accuracy 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 important 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.

	Accuracy	Detectability	Freshness	Reliability	Effort
whois IP	M/L	L	M	M	L
whois DNS	M/L	L	L	L	M/L
Routing Info	M	L	H	H	M/L
DNS names	M	L	M/H	L/M	M
GeoCluster	M	L	M	M/L	M
IDMaps*	L	L	L	M/L	M/H
GeoPing	H/M	H	H	H	M
CBG	H	H	H	H	M
GeoBud	H^+	H	H	H	H

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.

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

resource [wik06b]." 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.[4] 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.

[4] Wireless 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 encountering different network conditions increases. Compared to approaches based on information sources the results are very fresh.

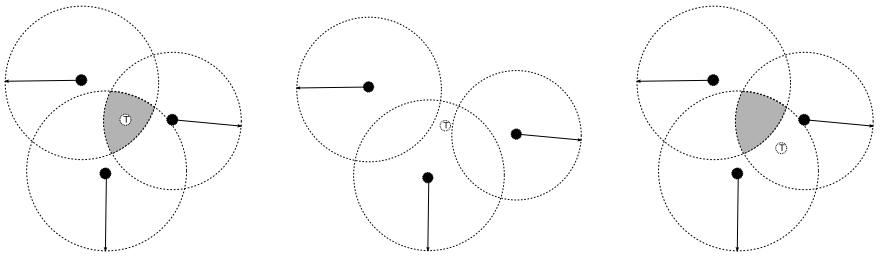
Using existing measurements between the landmarks are of course possible, this would result in a shorter time to complete a trace operation, but the measurements used would stretch over an even longer period of time than if doing the measurements asynchronously from the different landmarks.

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.



(a) Overestimated distance constraints.

(b) Underestimated distance constraints.

(c) Target mismatch.

Figure 3.4: The possible outcomes of varying the safety margin in CBG [15]. τ 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 overestimation 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 co-operation 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[7]. 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.

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

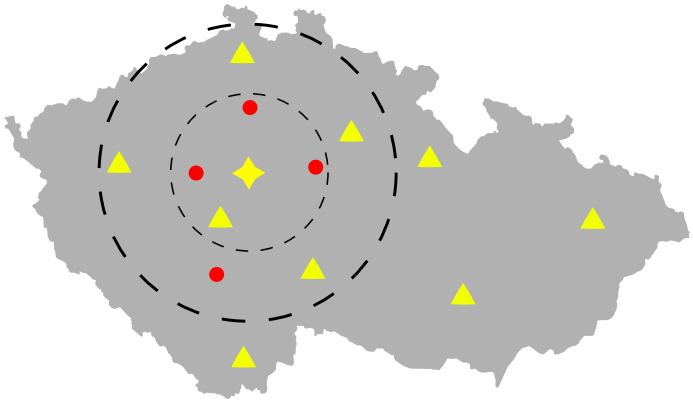


Figure 3.5: Selection of landmarks, shown as triangles, by the use of a super-landmark, shown here as a star. The dots symbolize population centers. The innermost circle is the delay from the super-landmark to the target, the outermost double this value. Landmarks inside the outermost circle are used 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 revealed 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[8]. 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.

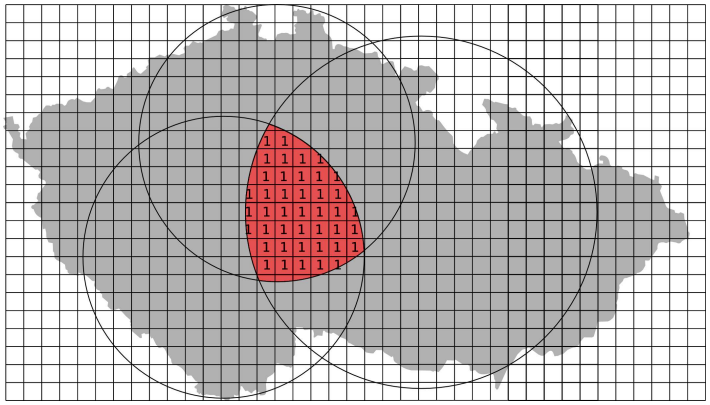


Figure 3.6: CBG converted to use grid coverage for constraint representation.

CBG using MPC

Implementing CBG using MPC immediately presents a problem. The algorithm for computing the confidence region $\mathcal{R}$ requires knowledge of all landmark locations[9]. 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

[8] It 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.

[9] The bestline 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.

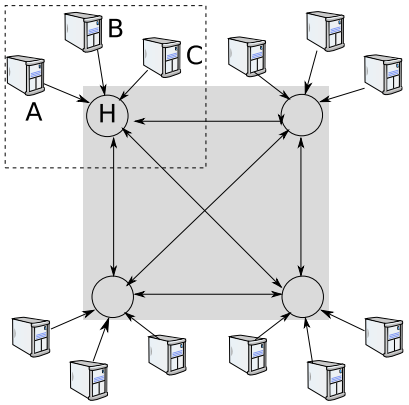


Figure 3.7: A possible configuration of limited MPC. Measurement nodes A, B and C form a trust cluster with H as the TTP. MPC is used between the information exchange hubs inside the grey area.

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 do so either. Further they 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[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.

$$d = d_t + d_p + q + \varepsilon \tag{3.1}$$

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

[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 ε is time wasted due to media access contention, router processing overhead, ARP[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 ε is known as stochastic or variable delay [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 ε[12]. Estimating the value of ε and whether or not the probe is delayed due to queueing is therefore important [39, 40].

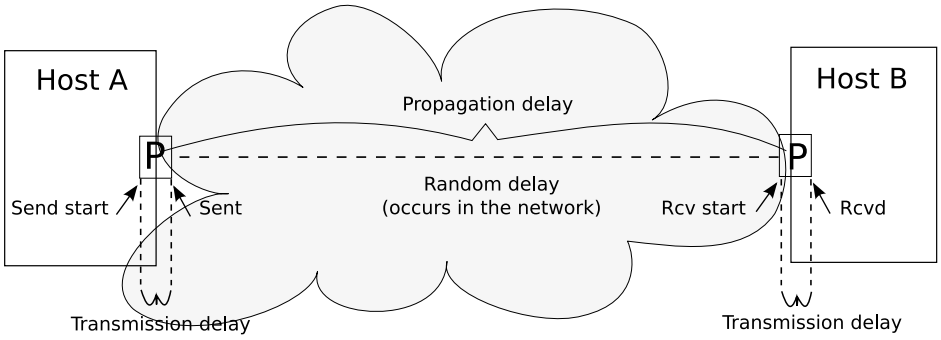


Figure 3.8: The different components that make up network delay.

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[13] packet switched network[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[15], due

[11] Address Resolution Protocol. Used to find the MAC address from the IP address.

[12] In [RFC2330], Framework for IP Performance Metrics, the term "wire speed" is used for the combination of d_p and ε.

[13] This might be about to change, as recent debate over net neutrality seems inclined towards service differentiation [net06].

[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.

[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[16]. [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[17] 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[18] 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[19] 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 []. 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 ε 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 ε 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 ε.
- 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 ε are used. The minimum RTT determines the bottom left corner of Region C1, see Figure 3.9. ε determines the boundaries for C1, C2 and C3. On a congestion-free path most probes should be within Region C1.

The mode[20] 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 ε using the observed minimum RTT and the mode RTT. ε 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 $RTT_i \leq (minRTT + \varepsilon)$ and $RTT_{i+1} \leq (minRTT + \varepsilon)$. It is part of C2 if $max(RTT_i, RTT_{i+1}) > (minRTT + \varepsilon)$ and $min(RTT_i, RTT_{i+1}) \leq (minRTT + \varepsilon)$. And finally it is part of C3 if $RTT_i > (minRTT + \varepsilon)$ and $RTT_{i+1} > (minRTT + \varepsilon)$.

[20] The most frequent values in a data sample is known as the mode of the sample [sta06].

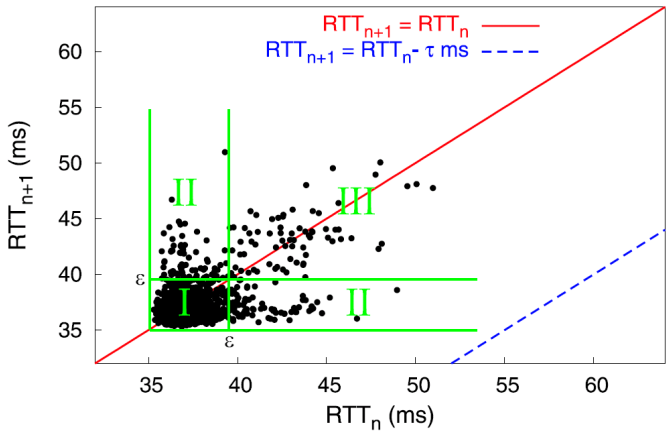


Figure 3.9: A phase plot of a RTT sample showing the distribution among the three congestion regions [40]. τ denotes the inter-probe delay.

Computing the Confidence

The confidence is expressed as $C1 + C2 + C3 = 1$. With N probe pairs the value of C1 is computed as in Equation 3.2,

$$C1 = \frac{1}{N} \sum_{p_i, i=1}^{N} \frac{1}{\Delta(RTT_i)} \times \frac{1}{\Delta(RTT_{i+1})}, \text{where}$$
$$\Delta(x) = \begin{cases} 1 & x = minRTT \\ \lceil \frac{x - minRTT}{\varepsilon} \rceil & x > minRTT \end{cases} \tag{3.2}$$

C2 is computed as in Equation 3.3.

$$C2 = \frac{1}{N} \left[K - \sum_{p_i \epsilon C2} \frac{1}{\Delta\left(max\left(RTT_i, RTT_{i+1}\right)\right)} \right], \tag{3.3}$$

where K is the number of probepairs in C2.

And C3 is computed as in Equation 3.4.

$$C3 = \frac{1}{N} \left[M - \sum_{p_i \epsilon C3} \frac{1}{\Delta(max(RTT_i, RTT_{i+1}))} \times \frac{1}{\Delta(RTT_{i+1})} \right], \tag{3.4}$$

where M is the number of probepairs in C3.

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 ε 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 x 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[21]. 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[22]-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[23] 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[24] 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

[21] Unique addresses in IPv4: 4,294,967,296.
Unique addresses in IPv6: 340,282,366,920,938,463,463,374,607,431,768,211,456

[22] Dynamic Host Control Protocol. Defined in [RFC2131]

[23] Media Access Control. (Globally) unique equipment identifiers used in many communication networks for identification at layer 2 in the OSI network stack.

[24] In 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 rec` by the HA to the MN by use of tunnelling. The reply `Rep` is sent directly from the MN's CoA to the CN.

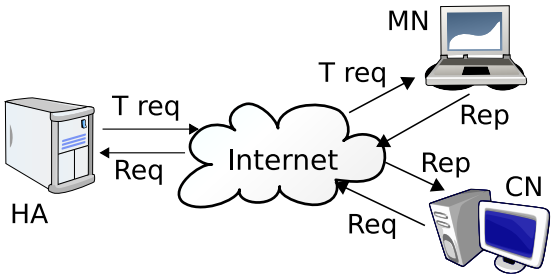


Figure 3.10: Triangle routing in Mobile IP.

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[25] 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.

Chapter 4

Geolocation in MAFIF

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.

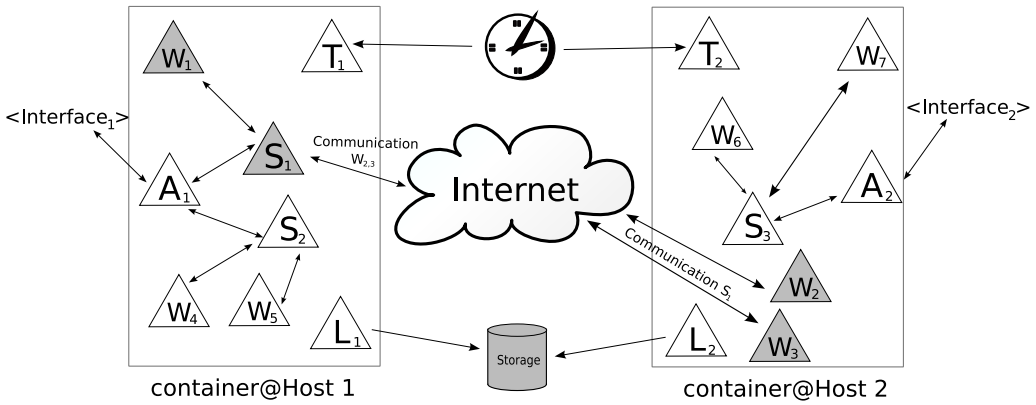


Figure 4.1: A High-Level Design of MAFIF, showing the different agents.

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[1].

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].

[1]The 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.

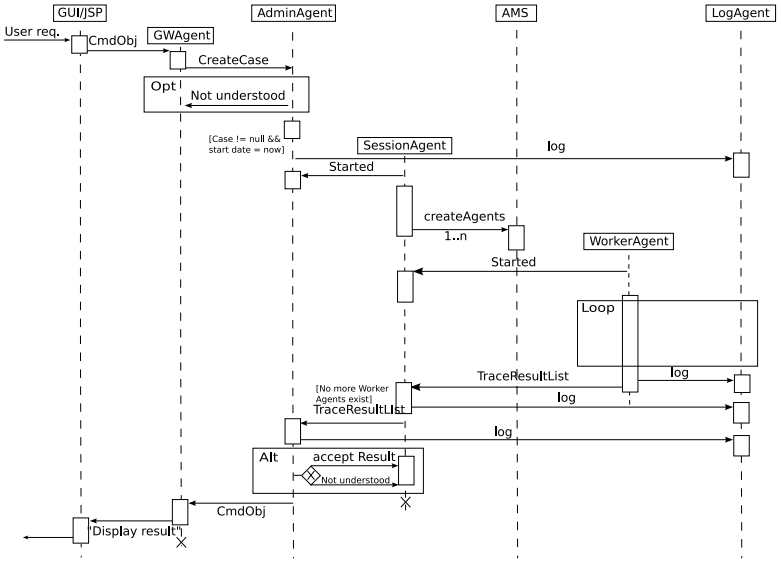


Figure 4.2: An AUML sequence diagram showing the stages of a geolocation operation, the entities involved and their actions.

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.

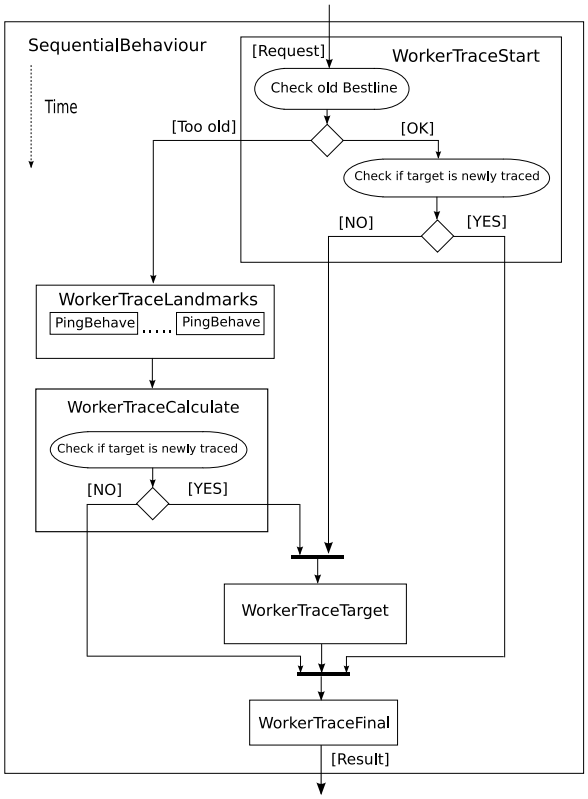


Figure 4.3: All the different WorkerTrace behaviours are sub-behaviours of a SequentialBehaviour such that they are executed in a linear sequence. The WorkerTraceLandmarks behaviour extends ParallelBehaviour and utilizes a ThreadedBehaviourFactory such that its PingBehave sub-behaviours are executed in separate threads in parallel.

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.

$$y - \frac{d_{ij} - b_i}{g_{ij}} x - b_i \geq 0, \forall_{j \neq i} \tag{4.1}$$

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,

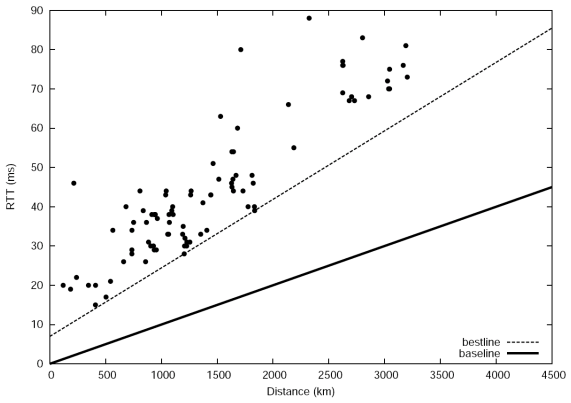


Figure 4.4: A scatter plot showing the relation between delay and geographical distance [15].

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.

$$m_i = \frac{d_{ij} - b_i}{g_{ij}} \tag{4.2}$$

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.

$$\min_{\substack{b_i \geq 0 \\ m_i \geq m}} \left(\sum_{i \neq j} y - \frac{d_{ij} - b_i}{g_{ij}} x - b_i \right) \tag{4.3}$$

The geographical constraint $\hat{g}_{i\tau}$ from each landmark to the target τ is calculated by Equation 4.4. This constraint actually represents a circle $\mathcal{C}_{i\tau}$ 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.

$$\hat{g}_{i\tau} = \frac{d_{i\tau} - b_i}{m_i} \tag{4.4}$$

The final step in CBG is to take the intersection of all the $\mathcal{C}_{i\tau}$'s defined by the geographical constraints and radii of the landmarks, and compute its area and

center. The region $\mathcal{R}$ intersected by all $\mathcal{C}_{i\tau}$ is the confidence region of the target τ's location. This intersection is defined as in Equation 4.5, where K is the total number of landmarks.

$$\mathcal{R} = \bigcap_{i=1}^{K} \mathcal{C}_{i\tau} \tag{4.5}$$

Performing Equation 4.5 requires one to find all points of intersection between the $\mathcal{C}_{i\tau}$'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 $\mathcal{C}_{i\tau}$'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.

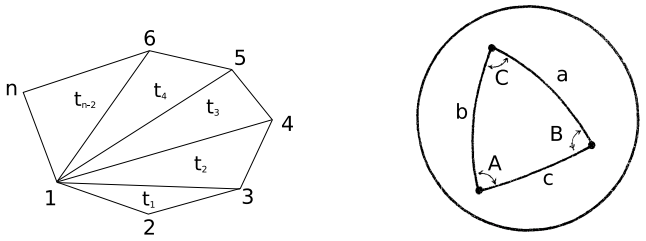
$$A_p = R^2 \left[\sum \alpha_i - (n-2)\pi \right] \tag{4.6}$$

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



(a) Splitting a polygon into triangles

(b) Angles in a spherical triangle

Figure 4.5: Triangles and angles of a polygon on a sphere.

$\mathcal{R}$ 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.

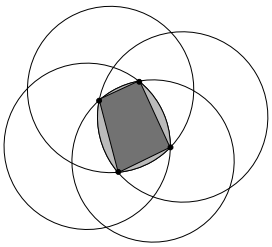


Figure 4.6: The difference between the area of a convex hull approximation, shown as the dark grey area, and the real area of an intersection of circles, shown as the combination of the dark and light gray areas combined.

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.

$$A_{\mathcal{R}} = \frac{1}{2} \sum_{n=0}^{N-1} \begin{vmatrix} x_n & x_{n+1} \\ y_n & Y_{n+1} \end{vmatrix} \tag{4.8}$$

$$c_x = \frac{1}{6A} \sum_{n=0}^{N-1} (x_n + x_{n+1}) \begin{vmatrix} x_n & x_{n+1} \\ y_n & Y_{n+1} \end{vmatrix} \tag{4.9}$$

$$c_y = \frac{1}{6A} \sum_{n=0}^{N-1} (y_n + y_{n+1}) \begin{vmatrix} x_n & x_{n+1} \\ y_n & Y_{n+1} \end{vmatrix} \tag{4.10}$$

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 $\mathcal{C}_{i\tau}$'s that lead to an empty intersection at all. It should be possible to discard up to a certain threshold of $\mathcal{C}_{i\tau}$'s if the remaining $\mathcal{C}_{i\tau}$ 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.

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

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.

```
1  From 213.242.106.46 icmp_seq=1 Packet filtered
2  From 213.242.106.46 icmp_seq=2 Packet filtered
3  From 213.242.106.46 icmp_seq=3 Packet filtered
4  From 213.242.106.46 icmp_seq=5 Packet filtered
5  From 213.242.106.46 icmp_seq=7 Packet filtered
6  From 213.242.106.46 icmp_seq=8 Packet filtered
7
8  --- www.amazon.co.uk ping statistics ---

9  9 packets transmitted, 0 received, +6 errors, 100% packet loss, time 8031ms
```

(a) Filtered ping output from www.amazon.co.uk.

```
1  64 bytes from semper16.itea.ntnu.no (129.241.56.206): icmp_seq=1 ttl=62 time=0.357 ms
2  64 bytes from semper16.itea.ntnu.no (129.241.56.206): icmp_seq=2 ttl=62 time=0.366 ms
3  64 bytes from semper16.itea.ntnu.no (129.241.56.206): icmp_seq=3 ttl=62 time=0.257 ms
4
5  --- semper16.itea.ntnu.no ping statistics ---
6  3 packets transmitted, 3 received, 0% packet loss, time 2009ms

7  rtt min/avg/max/mdev = 0.257/0.326/0.366/0.053 ms
```

(b) Normal ping output from semper16.itea.ntnu.no.

Figure 4.7: Ping outputs.

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 som 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 `HostUnreachableExcpetion` 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 ε values have been adapted from RTT-Ometer in [40]. The number of probes needed to accurately measure C1 and ε 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.

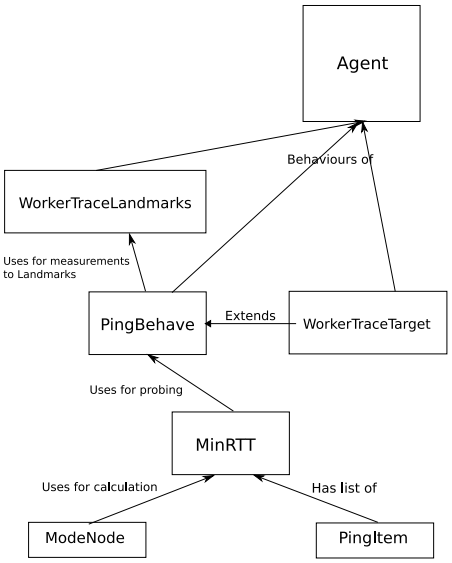


Figure 4.8: Relations of the Java classes concerned with RTT measurements.

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 ε

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.

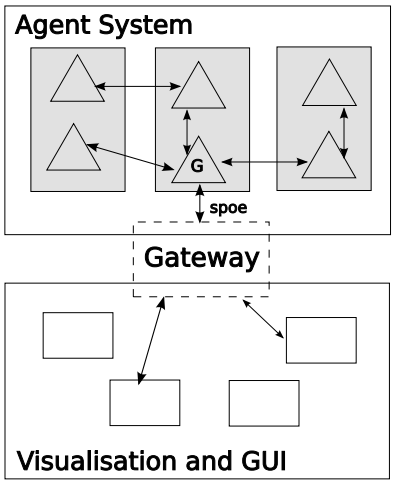


Figure 4.9: Information flow between the agent system and external parts. The gateway agent, denoted by a G, acts as a single point of entry into and out of the agent system.

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.

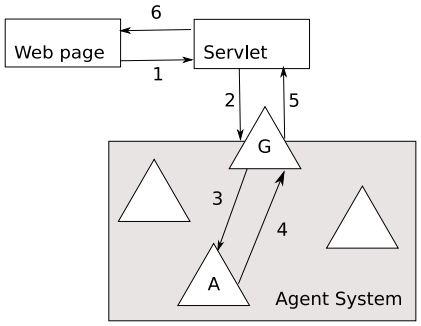


Figure 4.10: The sequence of events following a user-request to trace an address.

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-

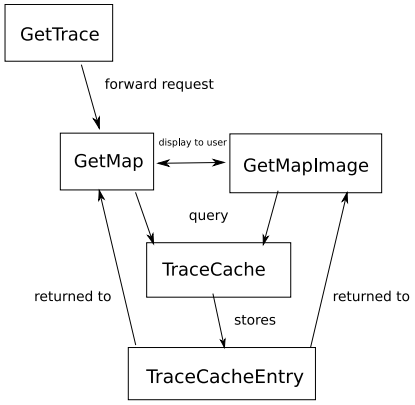


Figure 4.11: The relations between the servlet classes.

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[4] 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

[4] Version 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 a`java.awt.Shape`. Using methods defined by `java.awt.Area` which implements 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 unobtrusive 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.

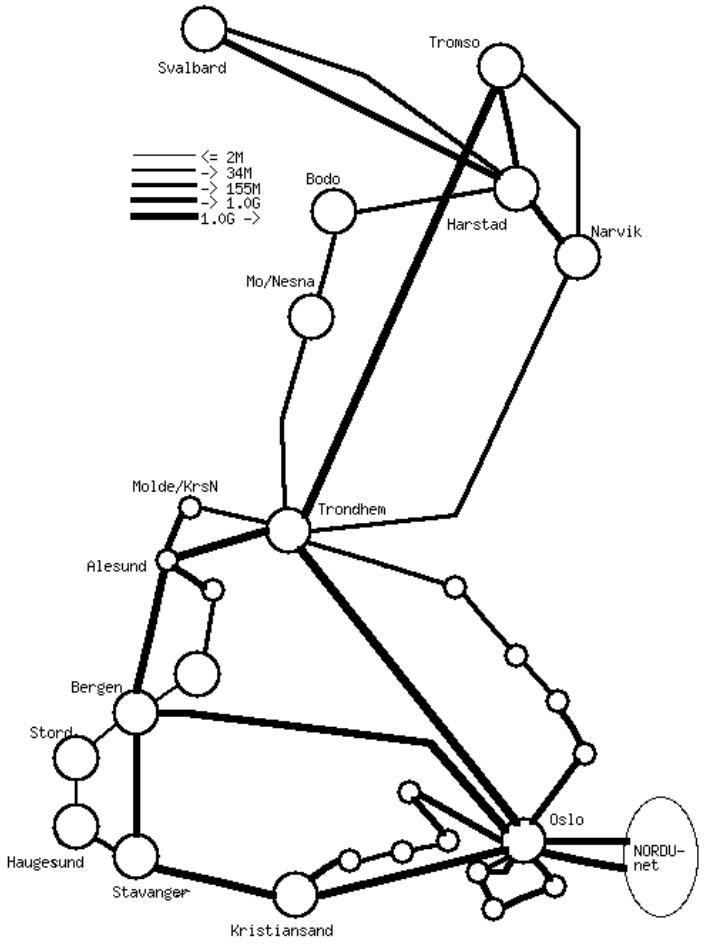


Figure 5.1: Logical topology of the Uninett network.

The geographic placement of the measurement nodes can be seen in Figure 5.2.[3] 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.

[3]The 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.

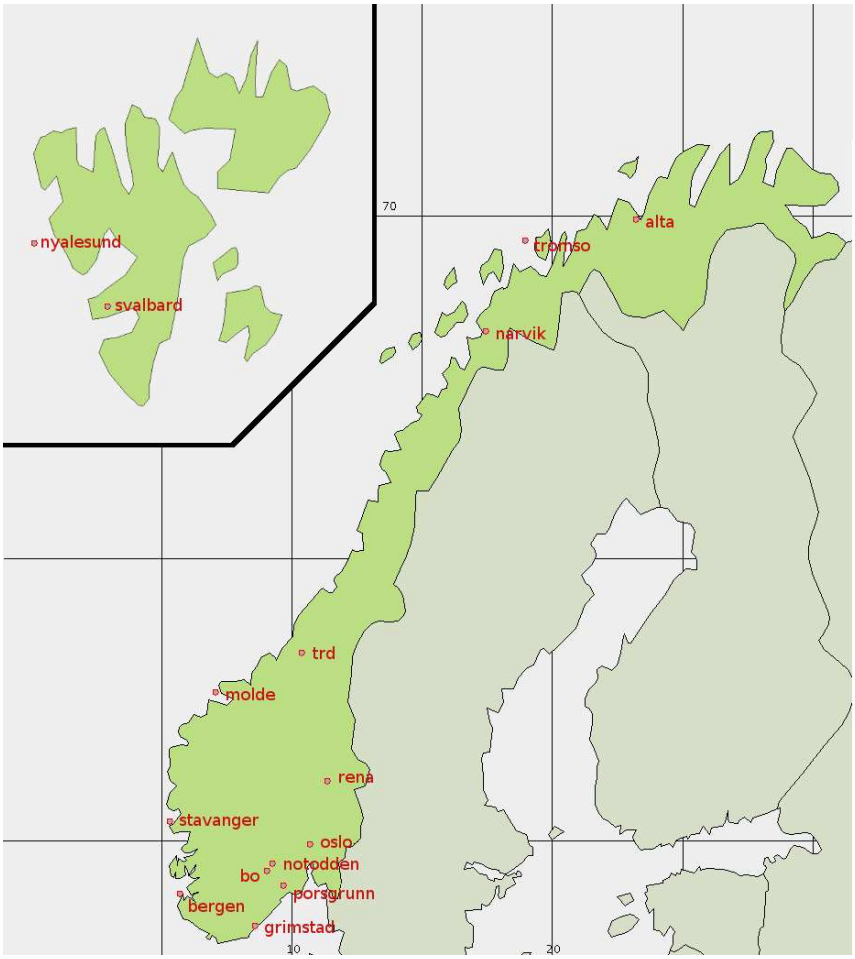


Figure 5.2: Placement of the 15 measurement nodes in the Uninett network.

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-

		bergen	bo	grimstad	molde	narvik	notodden	nyalesund	oslo	porsgrunn	rena	stavanger	svalbard	trd	tromso
alta	distance	1344.75	1345.38	1467.26	1073.51	284.88	1325.95	1028.07	1255.86	1361.17	1120.11	1472.99	919.94	918.38	166.37
	opt 1way	6.85	6.86	7.48	5.47	1.45	6.76	5.24	6.4	6.94	5.71	7.51	4.69	4.68	0.85
	1/2 rtt	14.1	14.11	15.7	12.57	3.7	13.47	10.75	13.34	13.7	11.69	17.14	11.48	9.49	1.49
bergen	distance		234.1	292.74	278.19	1064.96	236.66	2043.03	302.17	279.83	338.39	158.93	1985.64	429.59	1211.69
	opt 1way		1.19	1.49	1.42	5.43	1.21	10.41	1.54	1.43	1.72	0.81	10.12	2.19	6.18
	1/2 rtt		4.44	4.05	4.12	14.98	4.63	22.05	3.34	4.8	4.75	1.76	21.56	5.83	12.79
bo	distance	234.1		123.79	383.85	1081.41	19.74	2140.78	109.35	46.84	228.68	197.81	2072.52	449.86	1232.81
	opt 1way	1.19		0.63	1.96	5.51	0.1	10.91	0.56	0.24	1.17	1.01	10.56	2.29	6.28
	1/2 rtt	4.42		4.53	8.47	14.14	0.35	21.25	1.19	0.51	2.62	5.34	20.75	5.03	12.01
grimstad	distance	292.74	123.79		496.27	1204.7	142.43	2262.06	216.05	108.83	347.8	180.41	2194.83	573.34	1356.17
	opt 1way	1.49	0.63		2.53	6.14	0.73	11.53	1.1	0.55	1.77	0.92	11.19	2.92	6.91
	1/2 rtt	4.05	4.47		7.63	16.3	4.63	23.4	3.31	6.29	4.75	2.17	22.95	7.16	14.11
molde	distance	278.19	383.85	496.27		791.26	371.2	1775.94	366.29	424.16	286.28	426.21	1714.59	183.51	936.14
	opt 1way	1.42	1.96	2.53		4.03	1.89	9.05	1.87	2.16	1.46	2.17	8.74	0.94	4.77
	1/2 rtt	4.08	8.57	8.14		13.41	8.77	20.46	7.37	8.87	8.81	5.87	20.04	4.3	11.25
narvik	distance	1064.96	1081.41	1204.7	791.26		1062.37	1151.97	998.56	1101.21	861.49	1197.8	1059.74	643	151.59
	opt 1way	5.43	5.51	6.14	4.03		5.41	5.87	5.09	5.61	4.39	6.11	5.4	3.28	0.77
	1/2 rtt	14.92	14.14	16.51	13.32		14.35	8.51	12.99	14.49	12.5	16.61	7.99	9.16	2.37
notodden	distance	236.66	19.74	142.43	371.2	1062.37		2124.16	91.6	53.49	208.94	212.72	2055.41	432.08	1213.81
	opt 1way	1.21	0.1	0.73	1.89	5.41		10.83	0.47	0.27	1.06	1.08	10.48	2.2	6.19
	1/2 rtt	4.63	0.35	4.74	8.68	14.36		21.45	1.4	0.7	2.82	5.54	20.94	5.23	12.2
nyalesund	distance	2043.03	2140.78	2262.06	1775.94	1151.97	2124.16		2081.7	2171.88	1949.18	2198.98	120.07	1695.5	1023.45
	opt 1way	10.41	10.91	11.53	9.05	5.87	10.83		10.61	11.07	9.93	11.21	0.61	8.64	5.22
	1/2 rtt	22.06	21.27	23.53	20.48	8.52	21.48		20.13	21.64	19.62	23.74	0.61	16.3	9.51
oslo	distance	302.17	109.35	216.05	366.29	998.56	91.6	2081.7		107.29	137.12	303.21	2009.12	386.67	1150.13
	opt 1way	1.54	0.56	1.1	1.87	5.09	0.47	10.61		0.55	0.7	1.55	10.24	1.97	5.86
	1/2 rtt	3.34	1.19	3.43	7.34	13.02	1.39	20.11		1.56	1.51	4.22	19.61	3.91	10.86
porsgrunn	distance	279.83	46.84	108.83	424.16	1101.21	53.49	2171.88	107.29		241.06	227.42	2101.94	477.54	1252.78
	opt 1way	1.43	0.24	0.55	2.16	5.61	0.27	11.07	0.55		1.23	1.16	10.71	2.43	6.39
	1/2 rtt	4.79	0.51	6.29	8.85	14.52	0.71	21.64	1.56		2.98	8.05	21.11	5.4	12.36
	rtt	9.57	1.02	12.59	17.7	29.04	1.41	43.27	3.13		5.96	16.11	42.23	10.81	24.72
rena	distance	338.39	228.68	347.8	286.28	861.49	208.94	1949.18	137.12	241.06		395.77	1875.04	258.62	1013.06
	opt 1way	1.72	1.17	1.77	1.46	4.39	1.06	9.93	0.7	1.23		2.02	9.56	1.32	5.16
	1/2 rtt	4.74	2.62	4.87	8.79	12.48	2.81	19.59	1.51	2.97		5.65	19.1	3.36	10.32
	rtt	9.49	5.24	9.75	17.59	24.97	5.63	39.18	3.03	5.95		11.3	38.19	6.71	20.65
stavanger	distance	158.93	197.81	180.41	426.21	1197.8	212.72	2198.98	303.21	227.42	395.77		2139.82	555.08	1346.87
	opt 1way	0.81	1.01	0.92	2.17	6.11	1.08	11.21	1.55	1.16	2.02		10.91	2.83	6.86
	1/2 rtt	1.76	5.33	2.26	5.8	16.62	5.53	23.71	4.23	8.11	5.64		23.21	7.49	14.43
	rtt	3.51	10.66	4.52	11.6	33.23	11.06	47.42	8.46	16.23	11.28		46.42	14.97	28.86
svalbard	distance	1985.64	2072.52	2194.83	1714.59	1059.74	2055.41	120.07	2009.12	2101.94	1875.04	2139.82		1624.47	925.83
	opt 1way	10.12	10.56	11.19	8.74	5.4	10.48	0.61	10.24	10.71	9.56	10.91		8.28	4.72
	1/2 rtt	21.53	20.75	22.98	19.94	7.99	20.95	0.61	19.59	21.1	19.1	23.22		15.77	8.97
trd	distance	429.59	449.86	573.34	183.51	643	432.08	1695.5	386.67	477.54	258.62	555.08	1624.47		792.74
	opt 1way	2.19	2.29	2.92	0.94	3.28	2.2	8.64	1.97	2.43	1.32	2.83	8.28		4.04
	1/2 rtt	5.84	5.04	7.29	4.23	9.18	5.24	16.29	3.91	5.4	3.35	7.49	15.78		7.02
tromso	distance	1211.69	1232.81	1356.17	936.14	151.59	1213.81	1023.45	1150.13	1252.78	1013.06	1346.87	925.83	792.74	
	opt 1way	6.18	6.28	6.91	4.77	0.77	6.19	5.22	5.86	6.39	5.16	6.86	4.72	4.04	
	1/2 rtt	12.8	11.99	14.24	11.17	2.37	12.2	9.49	10.85	12.35	10.33	14.44	8.96	7	

Figure 5.3: The lower limit one-way delay, halved RTT and great circle distance between all measurement nodes.

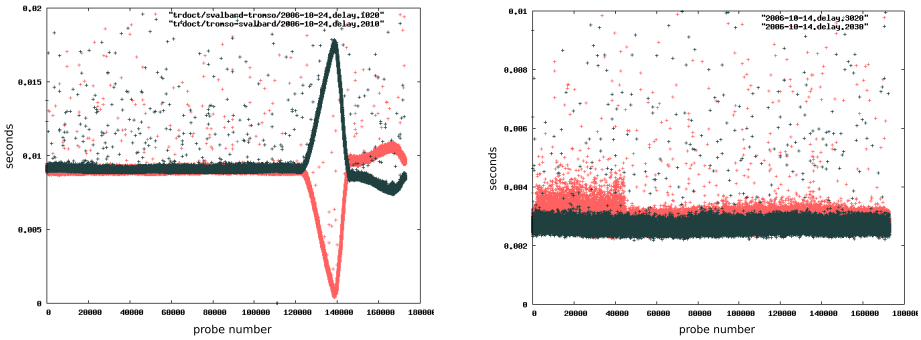
mp and porsgrunn-mp where halved RTT is larger by a factor of 11.35. The difference for narvik-mp - svalbard-mp is the smallest with a factor of 1.48. This seems reasonable as the cable between Narvik and Svalbard runs mostly along the ocean floor and thus probably is relatively close to the great circle 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.



(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).

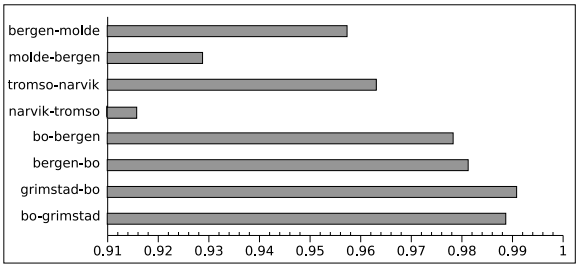


Figure 5.5: Correlation between halved minimum RTT and average one-way delay.

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[4]. 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

[4]The 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 the locate any host in a real forensic situation. The sections below describe limitations particular to our use of Uninett as the environment four 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

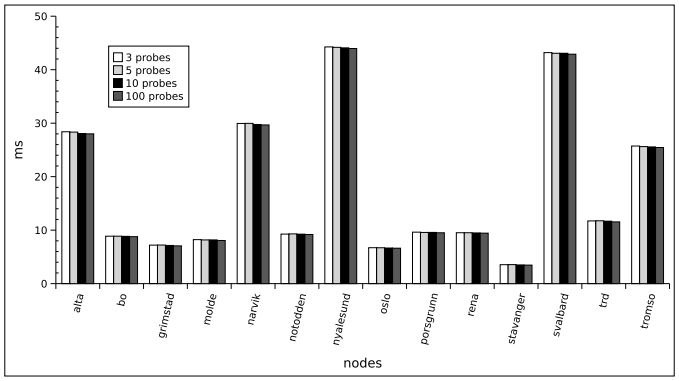
For the measurement based approaches the accuracy of the delay measurements are very decisive for the final results. The number of probes and the distance between

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?

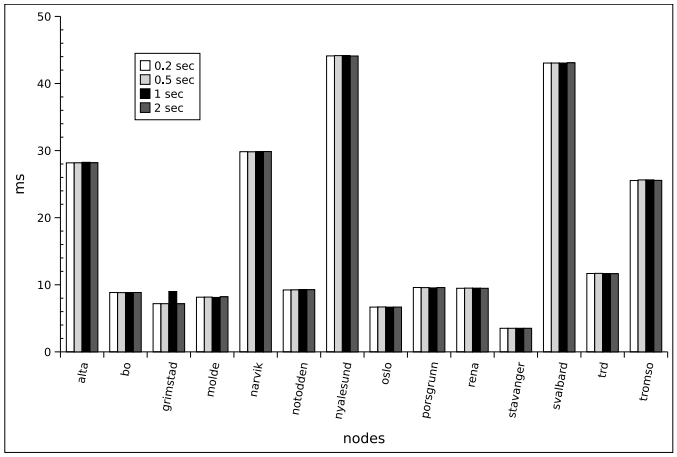
All combinations of the following values have been tested. Number of probes: 3, 5, 10 and 100. Distance between probes: 0.2, 0.5, 1 and 2 seconds. By using the techniques described in Section 3.6.3 we analyze the different combinations.

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.



(a) RTT comparison with varied number of probes.

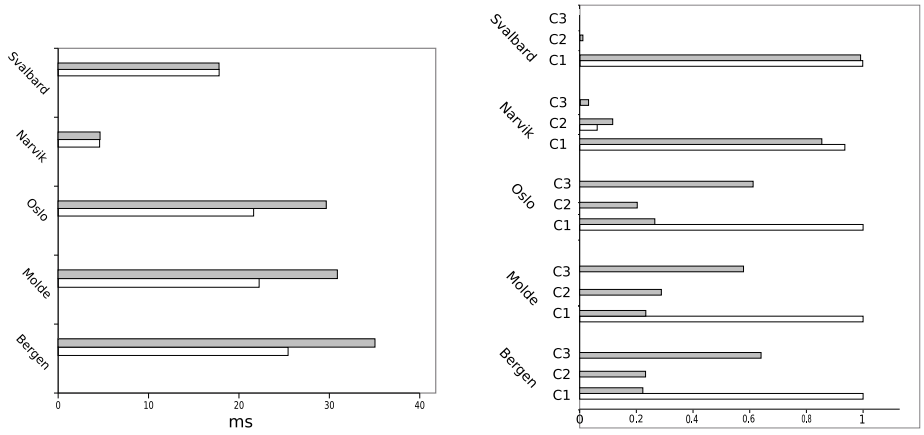


(b) Underestimated distance constraints.

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 Tromso 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-Bodø-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.



(a) Comparison of minimum RTT values.

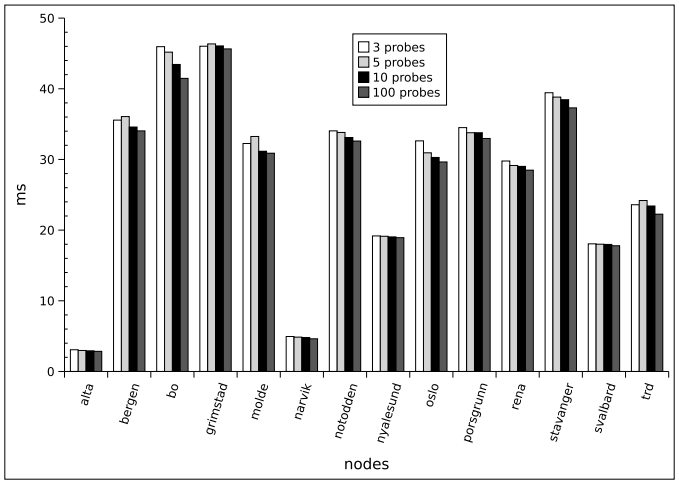
(b) Comparison of C values.

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

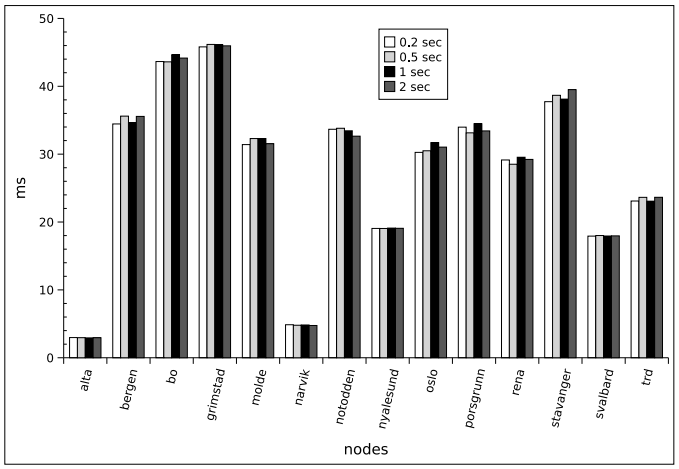
It is clear from Figure 5.7(a) that the downed link caused a partitioning of the Uninett network with regard to delay values and congestion. Hosts to which probes normally are routed along the Tromsø-Trondheim link show markedly increased delays. The increased C2 and especially C3 values in Figure 5.7(b) 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]. ε 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.

ε 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
IPv6 address 2001:700:300:11c0:20f:1fff:fe73:e2a8.

		trondeim-mp	narvik-mp	localhost
trd-mp	minRTT	X	18.33 / 19.48	0.29 / 0.76
	ε	X	0.70 / 0.88	-
narvik-mp	minRTT	18.33 / 19.55	X	18.47 / 20.05
	ε	0.78 / 0.82	X	0.73 / 0.82
localhost	minRTT	0.28 / 0.76	18.46 / 19.85	X
	ε	-	0.67 / 0.92	X

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

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[6].

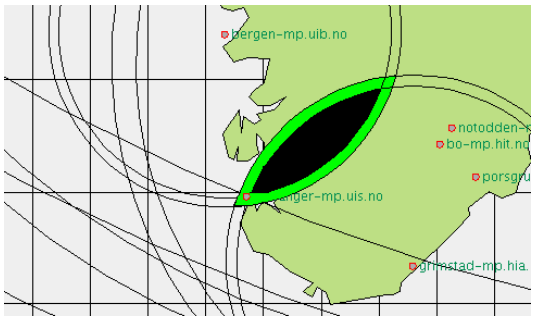


Figure 5.9: The confidence region of an actual geolocation of stavanger-mp using IPv4, shown inside the larger confidence region of an extrapolated IPv6 run.

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

[6] 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.

Target	Estimated Location	$\Delta(DV)$
narvik-mp	tromso-mp	11.9227
trd-mp	rena-mp	17.2234
bo-mp	notodden-mp	1.3367
oslo-mp	rena-mp	6.8543

Table 5.2: GeoPing results using all measurement nodes

Target	Estimated Location	$\Delta(DV)$
trd-mp	molde-mp	7.5934
bo-mp	notodden-mp	0.9074
oslo-mp	rena-mp	5.1863

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 $\mathcal{C}_{i\tau}$ 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 $\mathcal{C}_{i\tau}$ of oslo-mp greatly decreases the confidence region.

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

Table 5.4: CBG results.

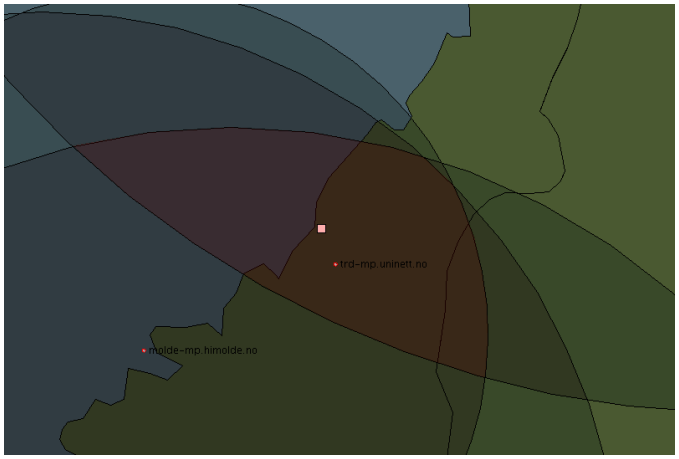


Figure 5.10: The best of the geolocation results using CBG.

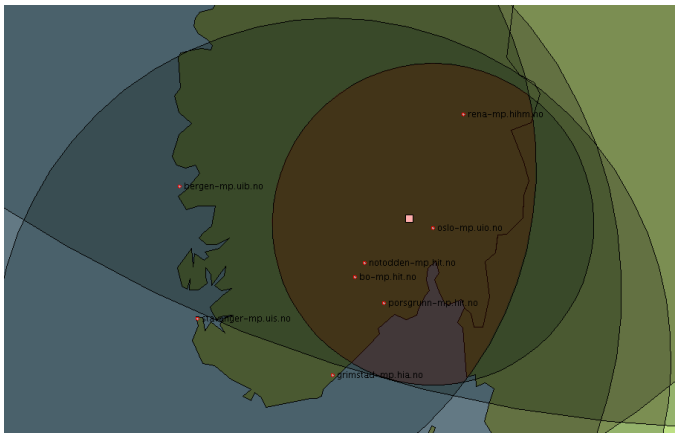


Figure 5.11: Confidence region and estimated location of bo-mp.

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

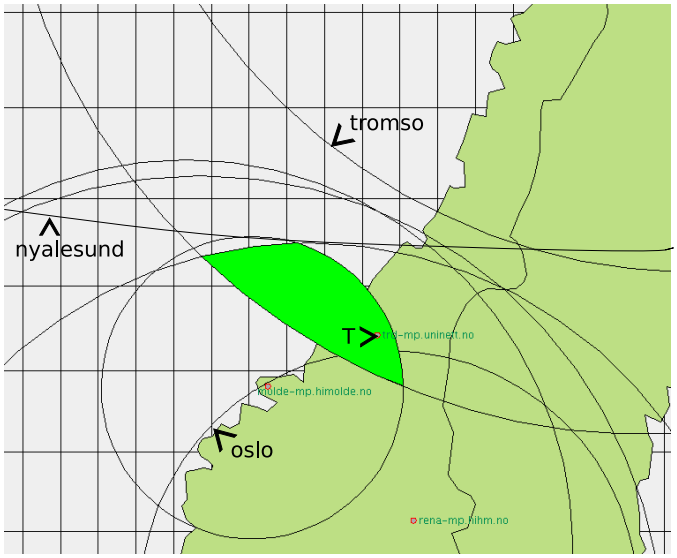
This experiment did not yield the expected results at all. Underestimations occurred frequently when using the standard factor of 2. Figure 5.12 gives two examples. Note that inclusion of the $\mathcal{C}_{i\tau}$ of oslo-mp in Figure 5.12(b) would not lead to an empty $\mathcal{R}$, but produce an incorrect 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 $\mathcal{C}_{i\tau}$s of landmarks that leads to an empty $\mathcal{R}$ should be implemented, as it would make CBG much more robust.

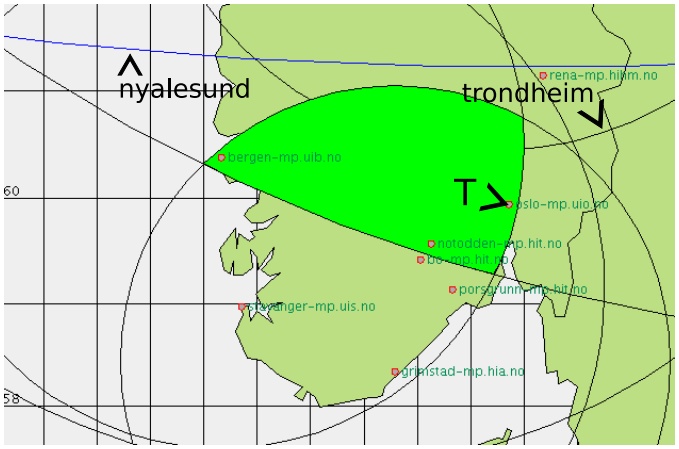
5.4.5 Moving Target

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.



(a) CBG run with trd-mp as the target T.



(b) CBG run with oslo-mp as the target T.

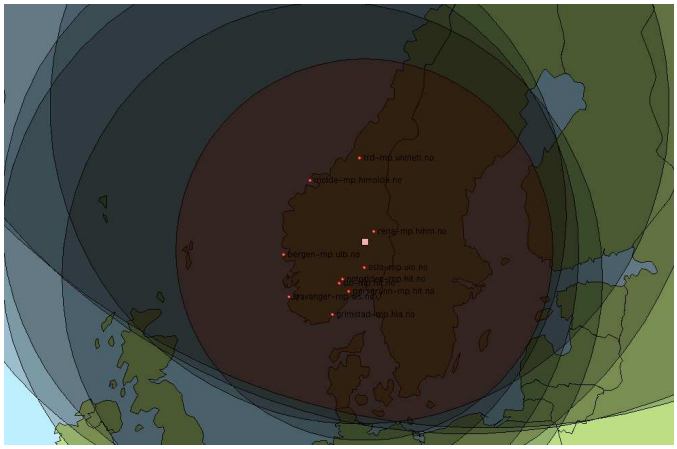
Figure 5.12: CBG runs with underestimations. Underestimated $\mathcal{C}_{i\tau}$s are indentified by arrows and landmark names.

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.

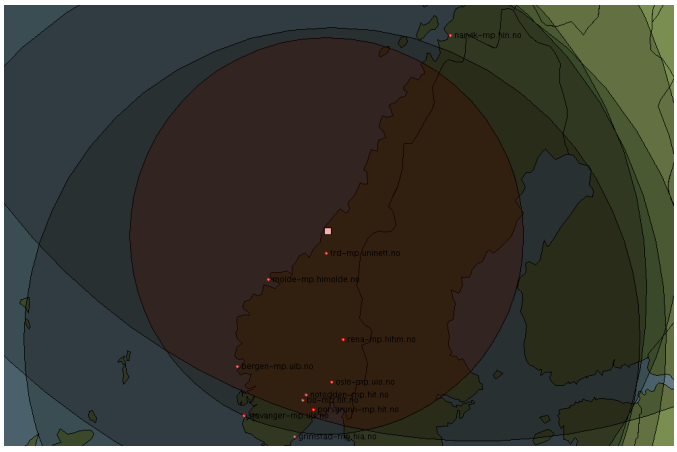
Results

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.



(a) CBG run for location Dragvoll. Estimated location: 60.7999, 10.7715. $\mathcal{R}$: 1 434 869 km^2.



(b) CBG run for location Nardo. Estimated location: 63.9739, 10.4898. $\mathcal{R}$: 927 349 km^2.

Figure 5.13: CBG runs wit target located at Nardo and Dragvoll.

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

	Estimated Location	min $\Delta(DV)$	max $\Delta(DV)$
Dragvoll			
	molde-mp	31.1318	32.1125
	porsgrunn-mp	32.3154	32.6606
Nardo			
	molde-mp	24.6418	30.0110
	porsgrunn-mp	30.4451	34.6249

Table 5.5: Moving target GeoPing results

It is clear from these results that the larger delays incurred by non-optical-fiber networks is relatively poorly handled by GeoPing and CBG.

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.

Estimated location	min$\Delta(DV)$	max $\Delta(DV)$
Nr of simultaneous instances: 1		
tromso-mp	10.75	-
svalbard-mp	39.56	-
trd-mp	44.86	-
Nr of simultaneous instances: 5		
tromso-mp	10.81	11.21
svalbard-mp	38.31	39.52
trd-mp	44.66	44.69
Nr of simultaneous instances: 10		
tromso-mp	10.76	11.83
svalbard-mp	39.14	39.65
trd-mp	44.56	45.72
Nr of simultaneous instances: 25		
tromso-mp	10.76	11.83
svalbard-mp	38.48	39.78
trd-mp	44.56	45.72

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

[1] Øystein E. Thorvaldsen. Internet investigations: Tools and methods. a multi-agent approach. Departement of Telematics, NTNU, May 2006.

[2] Espen André Fossen. Principles of Internet investigations: Basic reconnaissance, geopositionsing and public information sources. Master's thesis, Norwegian University of Science and Technology, June 2005.

[3] Espen André Fossen. Automatic tracing of Internet addresses. Departement of Telematics, NTNU, November 2004.

[4] Christian Larsen. Automatisk sikring av nettsted fra internett. Departement of Telematics, NTNU, November 2004.

[5] Gary Palmer. A road map for digital forensic research. Technical report, Digital Forensic Research Workshop, 2001.

[6] Espen André Fossen and André Årnes. Forensic geolocation of Internet addresses using network measurements. In *The 10th Nordic Workshop on Secure IT-systems (NORDSEC 2005)*, October 2005.

[7] Eoghan Casey. Error, uncertainty, and loss in digital evidence. *International Journal of Digital Evidence*, 1, Issue 2, 2002.

[8] J. Coppens, E.P. Markatos, J. Novotny, M. Polychronakis, V. Smotlacha, and S. Ubik. SCAMPI - a scaleable monitoring platform for the Internet. In *Proceedings of the 2nd International Workshop on Inter-Domain Performance and Simulation (IPS 2004)*, March 2004.

[9] J. A. Muir and P. C. van Oorschot. Internet geolocation and evasion. Technical Report TR 06-05, School of Computer Science, Carleton University, April 2006.

[10] Venkata N. Padmanabhan and Lakshminarayanan Subramanian. An investigation of geographic mapping techniques for internet hosts. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 173–185, New York, NY, USA, 2001. ACM Press.

[11] Artur Ziviani, Serge Fdida, José Ferreira de Rezende, and Otto Carlos Muniz Bandeira Duarte. Similarity models for Internet host location. In *Proc. of the IEEE International Conference on Networks - ICON'2003*, pages 81–86, Sydney, Australia, September 2003.

[12] Artur Ziviani, Serge Fdida, José Ferreira de Rezende, and Otto Carlos Muniz Bandeira Duarte. Toward a measurement-based geographic location service. In *Proc. of the Passive and Active Measurement Workshop - PAM'2004*, Lecture Notes in Computer Science (LNCS) 3015, pages 43–52, Antibes Juan-les-Pins, France, April 2004.

[13] Artur Ziviani, Serge Fdida, José Ferreira de Rezende, and Otto Carlos Muniz Bandeira Duarte. Demographic placement for Internet host location. In *Proc. of the IEEE GLOBECOM'2003*, volume 7, pages 3813–3817, San Francisco, CA, USA, December 2003.

[14] Bamba Gueye, Artur Ziviani, Serge Fdida, José F. de Rezende, and Otto Carlos M.B. Duarte. Two-tier geographic location of Internet hosts. *Lecture Notes in Computer Science*, 3079:730–739, January 2004.

[15] Bamba Gueye, Artur Ziviani, Mark Crovella, and Serge Fdida. Constraint-based geolocation of Internet hosts. In *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 288–293, 2004.

[16] Bamba Gueye, Steve Uhlig, Artur Ziviani, and Serge Fdida. Leveraging buffering delay estimation for geolocation of Internet hosts. *Lecture Notes in Computer Science*, 3976:319–330, January 2006.

[17] Paul Francis, Sugih Jamin, Vern Paxson, Lixia Zhang, Daniel F. Gryniewicz, and Yixin Jin. An architecture for a global Internet host distance estimation service. In *IEEE INFOCOM 1999*, pages 210–217, New York, NY, March 1999. IEEE.

[18] Paul Francis, Sugih Jamin, Cheng Jin, Yixin Jin, Danny Raz, Yuval Shavitt, and Lixia Zhang. IDMaps: a global Internet host distance estimation service. *IEEE/ACM Trans. Netw.*, 9(5):525–540, 2001.

[19] T. S. Eugene Ng and Hui Zhang. Towards global network positioning. In *IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pages 25–29, New York, NY, USA, 2001. ACM Press.

[20] T. S. E. Ng and Hui Zhang. Predicting Internet network distance with coordinates-based approaches. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 1, pages 170–179, 2002.

[21] Hyuk Lim, Jennifer C. Hou, and Chong-Ho Choi. Constructing Internet coordinate system based on delay measurement. In *IMC '03: Proceedings*

of the 3rd ACM SIGCOMM conference on Internet measurement, pages 129–142, New York, NY, USA, 2003. ACM Press.

[22] M. Pias, J. Crowcroft, S. Wilbur, S. Bhatti, and T. Harris. Lighthouses for scalable distributed location. In *Proc of Second International Workshop on Peer-to-Peer Systems (IPTPS '03)*, Berkeley, CA, USA, February 2003.

[23] Liying Tang and Mark Crovella. Virtual landmarks for the Internet. In *IMC '03: Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 143–152, New York, NY, USA, 2003. ACM Press.

[24] S. van Langen, X. Zhou, and P. Van Mieghem. On the estimation of Internet distances using landmarks. In *International Conference on Next Generation Teletraffic and Wired/Wireless Advanced Networking (NEW2AN'04)*, February 2004.

[25] X. Zhou and P. Van Mieghem. On the aging of landmark-based co-ordinates. In *IEEE joint conference of 10th Asia-Pacific Conference on Communications (APCC2004) and 5th International Symposium on Multi-Dimensional Mobile Communications (MDMC2004) (APCC/MDMC'04)*, pages 347–350, Beijing, China, August 2004.

[26] Michal Szymaniak, Guillaume Pierre, and Maarten van Steen. Scalable cooperative latency estimation. In *ICPADS '04: Proceedings of the Parallel and Distributed Systems, Tenth International Conference on (ICPADS'04)*, pages 367–376, Washington, DC, USA, July 2004. IEEE Computer Society.

[27] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 4(2), February 1981.

[28] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, August 2004.

[29] A. Yao. Protocols for secure computation. In *Proceedings of 23rd IEEE Symposium on the Foundations of Computer Science*, pages 160–166. IEEE, November 1982.

[30] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 1–10, 1988.

[31] David Chaum, Claude Crépeau, and Ivan Damgard. Multiparty unconditionally secure protocols. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 11–19, 1988.

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

equality, comparison, bits and exponentiation. In *Proceedings of the third Theory of Cryptography Conference TCC 2006*, pages 285–304, March 2006.

[33] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC '87: Proceedings of the nineteenth annual ACM conference on Theory of computing*, pages 218–229, New York, NY, USA, 1987.

[34] D. Beaver and S. Goldwasser. Multiparty computation with faulty majority. In *30th Annual Symposium on Foundations of Computer Science*, pages 468–473, November 1989.

[35] Juan A. Garay, Philip MacKenzie, and Ke Yang. Efficient and secure multi-party computation with faulty majority and complete fairness. Cryptology ePrint Archive, Report 2004/009, 2004.

[36] Tord Ingolf Reistad. Multi-party secure position determination. In *Norsk Informatikk Konferanse 2006*, June 2006.

[37] Headquarters Department of the Army, Washington, USA. *Field Manual Map Reading and Land Navigation*, no. 3-25.26 edition, July 2001.

[38] Omer Gurewitz, Israel Cidon, and Moshe Sidi. One-way delay estimation using network-wide measurements. *IEEE Transactions on Information Theory*, 52(6):2710–2724, 2006.

[39] Z. Wang, A. Zeitoun, and S. Jamin. Challenges and lessons learned in measuring path RTT for proximity-based applications. In *The Passive and Active Measurement Workshop (PAM2003)*. The NLANR Measurement and Network Analysis Group, April 2003.

[40] Amgad Zeitoun, Zhiheng Wang, and Sugih Jamin. RTTometer: Measuring path minimum RTT with confidence. In *IEEE Workshop on IP Operations and Management (IPOM 2003)*, pages 127–134, October 2003.

[41] V. Paxson. End-to-end routing behaviour in the internet. In *Proceedings of ACM SIGCOMM'96*, pages 25–38, August 1996.

[42] Y. Zhang, V. Paxson, and S. Shenker. The stationarity of internet path properties: Routing, loss, and throughput. *ACIRI Technical Report*, 2000.

[43] Vern Paxson. End-to-end Internet packet dynamics. In *SIGCOMM '97: Proceedings of the ACM SIGCOMM '97 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 139–152, 1997.

[44] Chuck Fraleigh, Sue Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and Christophe Diot. Packet-level traffic measurements from the Sprint IP backbone. *IEEE Network*, 17(6):6–16, 2003.

[45] Tin Yu Wu, Han Chieh Chao, Tak Goa Tsuei, and Yu Feng Li. A measurement study of network efficency for TWAREN IPv6 backbone. *International Journal of Network Management*, 15:411–419, 2005.

[46] A. Zeitoun, C.-N. Chuah, S. Bhattacharyya, and C. Diot. An AS-level study of Internet path delay characteristics. In *Proceedings of IEEE Globecom*, Dallas, Texas, November 2004.

[47] Stefan Savage. Sting: A TCP-based network measurement tool. In *USENIX Symposium on Internet Technologies and Systems*, 1999.

[48] Omer Gurewitz, Israel Cidon, and Moshe Sidi. Network time synchronization using clock offset optimization. In *ICNP '03: Proceedings of the 11th IEEE International Conference on Network Protocols*, page 212, 2003.

[49] Darryl Veitch, Satish Babu, and Attila Pàsztor. Robust synchronization of software clocks across the Internet. In *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 219–232, 2004.

[50] Anurag Acharya and Joel Saltz. A study of internet round-trip delay. Technical Report CS-TR-3736, University of Maryland, 1996.

[51] John E. Midwinter. *Optical Fibres for Transmission*. John Wiley & Sons, New York, 1979.

[52] Lixin Gao and Feng Wang. The extent of AS path inflation by routing policies. In *Global Telecommunications Conference (GLOBECOM '02)*, volume 3, pages 2180–2184. IEEE, November 2002.

[53] Hongsuda Tangmunarunkit, Ramesh Govindan, and Scott Shenker. Internet path inflation due to policy routing. In Sonia Fahmy and Kihong Park, editors, *Proc. SPIE Scalability and Traffic Control in IP Networks*, volume 4526, pages 188–195. SPIE, July 2001.

[54] Neil Spring, Ratul Mahajan, and Thomas Anderson. Quantifying the causes of path inflation. ACM SIGCOMM, August 2003.

[55] Alberto Escudero Pascual. Location privacy in IPv6 "tracking the binding updates". In *Proceedings of IMDS01*, Lancaster, UK, September 2001.

[56] Tuomas Aura and Alf Zugenmaier. Privacy, control and Internet mobility. Presented at 12th International Workshop, Cambridge, UK To appear in Lecture Notes in Computer Science , Vol. 3957 September 2006, April 2004.

[57] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. Developing multi-agent systems with a FIPA-compliant agent framework. *Software: Practice and Experience*, 31(2), 2001.

[58] F. Bergenti, G. Caire, R. Pels, and C. van Aart. Creating and using ontologies in agent communication. *EXP in search of innovation*, 2(3):10–21, 2002.

[59] B. Bauer. Extending UML for the specification of interaction protocols. submission for the 6th call for Proposal of FIPA and revised version of FIPA 99, 1999.

[60] Bernhard Bauer. UML class diagrams revisited in the context of agent-based systems. In *AOSE '01: Revised Papers and Invited Contributions from the Second International Workshop on Agent-Oriented Software Engineering II*, pages 101–118, London, UK, 2002. Springer-Verlag.

[61] M.-P. Huget. Agent UML class diagrams revisited. In Bernhard Bauer, Klaus Fischer, Jorg Muller, and Bernhard Rumpe, editors, *Agent Technology and Software Engineering (AgeS)*, October 2002.

[62] Michael Bevis and Greg Cambareri. Computing the area of a spherical polygon of arbitrary shape. *Mathematical Geology*, 19(4):335–346, January 1987.

[63] Dominic Greenwood and Monique Calisti. An automatic bi-directional service integration gateway. In *Cybernetics and Man Conference*. IEEE, October 2004.

[64] Thang Xuan Nguyen and Ryszard Kowalczyk. Ws2jade: Integrating web service with jade agents. Technical Report SUTICT-TR2005.03, Swinburne University of Technology, July 2005.

[65] Keith Ryden. *OpenGIS Implementation Specification for Geographic information - Simple feature access - Part 2: SQL option*. Open Geospatial Consortium, 1.1.0 edition, November 2005.

[66] Sun Microsystems. *JDBC 4.0 API Specification - Proposed Final Draft*, July 2006.

[67] JADE Board. *Jade Administrator's Guide*. TILAB S.p.A, 3.3 edition, January 2005.

RFC References

[RFC1034] P. Mockapetris. Domain names - concepts and facilities, November 1987.

[RFC1035] P. Mockapetris. Domain names - implementation and specification, November 1987.

[RFC1305] David L. Mills. Network time protocol (version 3) specification, implementation and analysis, March 1992.

[RFC1712] C. Farrell, M. Schulze, S. Pleitner, and D. Baldoni. DNS encoding of geographical location, November 1994.

[RFC1876] C. Davis, P. Vixie, T. Goodwin, and I. Dickinson. A means for expressing location information in the domain name system, January 1996.

[RFC1918] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. Address allocation for private internets, February 1996.

[RFC1930] J. Hawkinson and T. Bates. Guidelines for creation, selection, and registration of an Autonomous System (AS), March 1996.

[RFC2050] K. Hubbard, M. Kosters, D. Conrad, D. Karrenberg, and J. Postel. Internet registry IP allocation guidelines, November 1996.

[RFC2131] R. Droms. Dynamic host configuration protocol, March 1997.

[RFC2330] V. Paxson, G. Almes, J. Mahdavi, and M. Mathis. Framework for IP performance metrics, May 1998.

[RFC2460] S. Deering and R. Hinden. Internet protocol, version 6 (IPv6) specification, December 1998.

[RFC2462] S. Thomson and T. Narten. IPv6 stateless address autoconfiguration, December 1998.

[RFC2622] C. Alaettinoglu, C. Villamizar, E. Gerich, D. Kessens, D. Meyer, T. Bates, D. Karrenberg, and M. Terpstra. Routing policy specification language (RPSL), June 1999.

[RFC3041] T. Narten and R. Draves. Privacy extensions for stateless address autoconfiguration in IPv6, January 2001.

[RFC3330] IANA. Special-use IPv4 addresses, September 2002.

[RFC3344] C. Perkins. IP mobility support for IPv4, August 2002.

[RFC3775] D. Johnson, C. Perkins, and J. Arkko. Mobility support in IPv6, June 2004.

[RFC3776] J. Arkko, V. Devarapalli, and F. Dupont. Using IPsec to protect mobile IPv6 signaling between mobile nodes and home agents, June 2004.

[RFC3912] L. Daigle. Whois protocol specification, September 2004.

[RFC4012] L. Blunk, J. Damas, F. Parent, and A. Robachevsky. Routing policy specification language next generation (RPSLng), March 2005.

[RFC4033] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. DNS security introduction and requirements, March 2005.

[RFC4140] H. Soliman, C. Castelluccia, K. El Malki, and L. Bellier. Hierarchical mobile IPv6 mobility management (HMIPv6), August 2005.

[RFC4271] Y. Rekhter, T. Li, and S. Hares. A border gateway protocol 4 (BGP-4), January 2006.

[RFC812] Ken Harrenstien and Vic White. Nicname/whois, March 1982.

[RFC882] P. Mockapetris. Domain names - concepts and facilities, November 1983.

[RFC883] P. Mockapetris. Domain names - implementation and specification, November 1983.

[RFC920] J. Postel and J. Reynolds. Domain requirements, October 1984.

[RFC954] K. Harrenstien, M. Stahl, and E. Feinler. Nicname/whois, October 1985.

Floating References

[aja06] http://goog-ajaxslt.sourceforge.net/, October 2006.

[ano06] Free haven's anonymity bibliography. http://freehaven.net/anonbib/, November 2006.

[ccW06] Wikipedia article on chain of custody. http://en.wikipedia.org/wiki/Chain_of_custody, September 2006.

[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.

[dns06a] Wikipedia article on DNSSEC. http://en.wikipedia.org/wiki/DNS_Security_Extensions, October 2006.

[dns06b] Wikipedia article on the domain name system. http://en.wikipedia.org/wiki/Domain_name_system, October 2006.

[Dut06] Steven Dutch. Converting UTM to latitude and longitude (or vice versa). http://www.uwgb.edu/dutchs/UsefulData/UTMFormulas.HTM, September 2006.

[geo06] GeoTools official homepage. http://geotools.codehaus.org/, July 2006.

[goo06a] Google maps homepage. http://maps.google.com, September 2006.

[goo06b] Wikipedia article on Google Earth. http://en.wikipedia.org/wiki/Google_Earth, September 2006.

[gre06] Wikipedia article on great circle distance. http://en.wikipedia.org/wiki/Great-circle_distance, September 2006.

[hij06a] CompleteWhois list of hijacked IPs. http://www.completewhois.com/hijacked/index.htm, October 2006.

[hij06b] CompleteWhois's questions and answers on IP hijacking. http://www.completewhois.com/hijacked/hijacked_qa.htm, October 2006.

[hsq06] HSQLDB official homepage. http://www.hsqldb.org, August 2006.

[IOC06] IOCE. G8 proposed principles for the procedures relating to digital evidence. http://www.ioce.org/G8_proposed_principles_for_forensic_evidence.html, February 2006.

[ipu06] iputils homepage. ftp://ftp.inr.ac.ru/ip-routing/iputils-current.tar.gz, October 2006.

[jad06] jademx - JMX access to Jade agents. http://jademx.sourceforge.net/, October 2006.

[jdb06] Sun's list of available JDBC drivers. http://developers.sun.com/product/jdbc/drivers, October 2006.

[jpc06] Jpcap - Java package for packet capture. http://netresearch.ics.uci.edu/kfujii/jpcap/doc/index.html, August 2006.

[jts06a] Jump Topology Suite official homepage. http://www.jump-project.org/project.php?PID=JTS\&SID=OVER, July 2006.

[jts06b] Wikipedia article on Jump Topology Suite (JTS). http://en.wikipedia.org/wiki/JTS_Topology_Suite, July 2006.

[kml06a] The official KML documentation. http://earth.google.com/kml/, September 2006.

[kml06b] Wikipedia article on Keyhole Markup Language. http://en.wikipedia.org/wiki/Keyhole_Markup_Language, September 2006.

[lan06] Landserf official homepage. http://www.landserf.org, July 2006.

[lob06] The large-scale monitoring of broadband internet infrastructure project LOBSTER. http://www.ist-lobster.eu/, October 2006.

[map06] Wikipedia article on map projections. http://en.wikipedia.org/wiki/Map_projection, September 2006.

[net06] Wikipedia article on network neutrality. http://en.wikipedia.org/wiki/Network_neutrality, November 2006.

[ope06a] Open Geospatial Consortium (OGC). http://www.opengeospatial.org/, July 2006.

[ope06b] Open source GIS. http://opensourcegis.org/, September 2006.

[ope06c] Openmap official homepage. http://www.openmap.org, July 2006.

[pin06] Wikipedia article on ping. http://en.wikipedia.org/wiki/Ping, November 2006.

[roc06] RockSaw API for raw socket network I/O in Java. http://www.savarese.org/software/rocksaw/index.html, August 2006.

[sca06] A scaleable monitoring platform for the internet (SCAMPI). http://www.ist-scampi.org/, July 2006.

[sta06] Wikipedia article on statistical mode. http://en.wikipedia.org/wiki/Mode_(statistics), September 2006.

[utm06] Wikipedia article on universal transverse mercator coordinate system. http://en.wikipedia.org/wiki/Universal_Transverse_Mercator_coordinate_system, September 2006.

[wgs06] Wikipedia article on world geodetic system. http://en.wikipedia.org/wiki/WGS84, September 2006.

[who06] Wikipedia article on whois service. http://en.wikipedia.org/wiki/WHOIS, October 2006.

[wik06a] Wikipedia article on network address translation. http://en.wikipedia.org/wiki/Network_address_translation, November 2006.

[wik06b] Wikipedia article on proxy server. http://en.wikipedia.org/wiki/Proxy_server, November 2006.

[wik06c] Wikipedia article on virtual private network. http://en.wikipedia.org/wiki/VPN, November 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

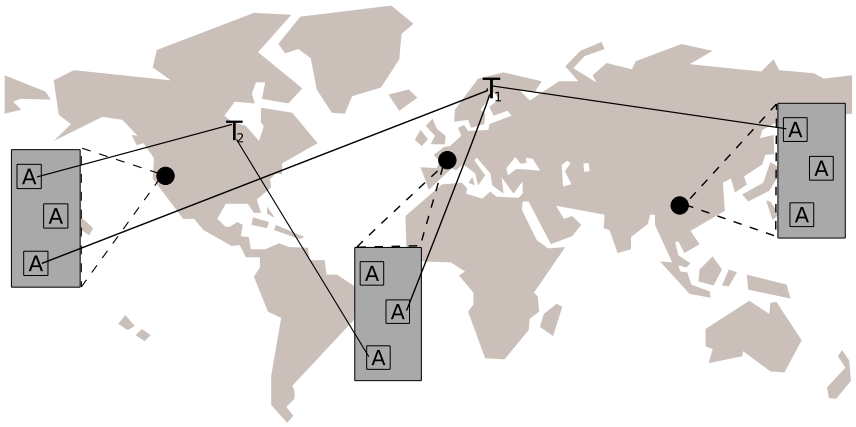


Figure 1: An example distribution of hosts participating in the system. A's are agents T_1 and T_2 are investigation targets.

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.

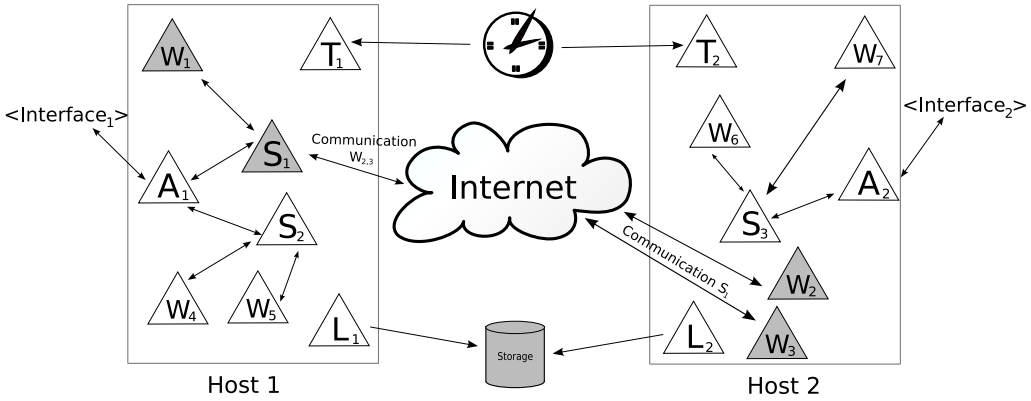


Figure 2: A High-Level Design of The Framework

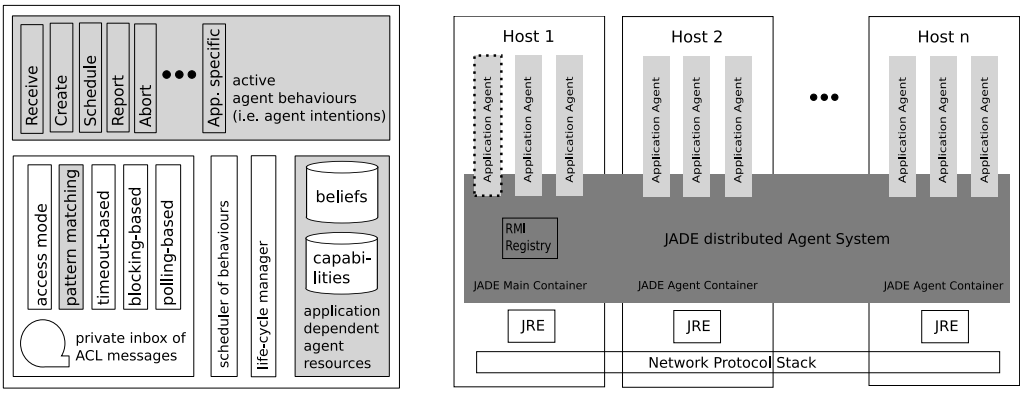
The AdminAgent agent type is the operator's single point of entry into the system. It is provided with several behaviours to handle incoming requests, schedule (future) executions and report on or abort any of its investigations, see Figure 3(a). The AdminAgent does not execute any investigations itself, 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

the same investigation session.

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 queryes 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.



(a) The architecture of a JADE agent. Light grey areas have been modified by us.

(b) A conceptual view of the JADE agent system, showing the distribution of agents among participating hosts.

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.

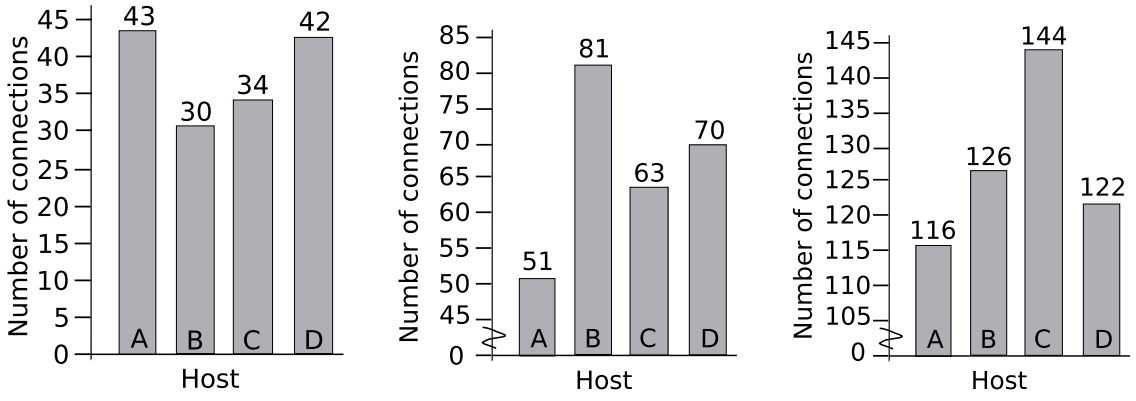
5 Experimental Results

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 refered 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.



(a) Simple (b) University (c) Newspaper

Figure 4: Number of connections to the webserver per host when running the different test cases.

As can be seen in Figure 4 no host exihibit 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

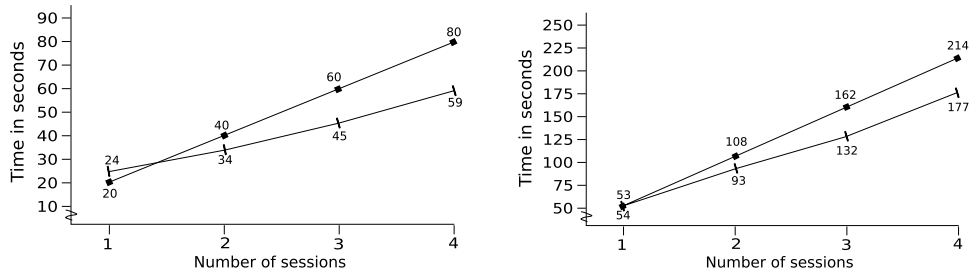
Load balancing among the hosts in the system is important to avoid any host acting as a bottleneck to the whole system. Scalability with regard to the number of active agents and the number of simultaneous investigation sessions is also important. This is a good indicator of the scalability of the framework,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

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

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-cautios 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.



(a) University case

(b) Newspaper case

Figure 5: Elapsed time for sets of sequential vs parallel test runs of content securing.

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 assosciated with agent life-cycle management for the agent system. The results in Figure 5, being from an unomptimized 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

[1] The Protégé Ontology Editor and Knowledge Acquisition System. `http://protege.stanford.edu`, accessed April 2006.

[2] J. S. Balasubramaniyan, J. O. Garcia-Fernandez, D. Isacoff, E. Spafford, and D. Zamboni. An architecture for intrusion detection using autonomous agents. In *Proceedings of the 14th Annual Computer Security Applications Conference*, pages 13–24. IEEE Computer Society, 1998.

[3] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. Developing multi-agent systems with a FIPA-compliant agent framework. *Software: Practice and Experience*, 31(2), 2001.

[4] John Bethencourt, Jason Franklin, and Mary Vernon. Mapping Internet sensors with probe response attacks. In *Proceedings of the 14th USENIX Security Symposium*, pages 193–208. USENIX, August 2005.

[5] Kalle Burbeck, Daniel Garpe, and Simin Nadjm-Tehrani. Scale-up and performance studies of three agent platforms. In *Proceedings of International Performance, Communication and Computing Conference, Middleware Performance workshop*, pages 857–863, April 2004.

[6] Krzysztof Chmiel, Maciej Gawinecki, Pavel Karczmarek, Michal Szymczak, and Marcin Paprzycki. Efficency of JADE agent platform. *Scientific Programming*, 13:159–172, 2005.

[7] M. Crosbie and E. Spafford. Defending a computer system using autonomous agents. In *Proceedings of the 8th National Information Systems Security Conference*, number 008 in 95, October 1995.

[8] M. Crosbie and G. Spafford. Active defense of a computer system using autonomous agents. Technical report, COAST Group, Dept. of Computer Sciences, Purdue University, February 1995.

[9] Espen André Fossen. Automatic tracing of Internet addresses. Departement of Telematics, NTNU, November 2004.

[10] Espen André Fossen. Principles of Internet investigations: Basic reconnaissance, geopositioning and public information sources. Master's thesis, Departement of Telematics, NTNU, July 2005.

[11] Espen André Fossen and André Årnes. Forensic geolocation of internet addresses using network measurments. In Helger Lipmaa and Dieter Gollman, editors, *Proceedings of the 10th Nordic Workshop on Secure IT Systems*, Tartu, Estonia, October 2005.

[12] G. Helmer, J. Wong, V. Honavar, and L. Miller. Lightweight agents for intrusion detection. *J. Syst. Softw.*, 67(2):109–122, 2003.

[13] Chet Hosmer. Proving the integrity of digital evidence with time. *International Journal of Digital Evidence*, 1, Issue 1, 2002.

[14] Curtis A. Carver Jr., John M.D. Hill, John R. Surdu, and Udo W. Pooch. A methodology for using intelligent agents to provide automated intrusion response. In *Proceedings of the IEEE Workshop on Information Assurance and Security*, 2000.

[15] Christian Larsen. Automatisk sikring av nettsted fra internett. Departement of Telematics, NTNU, November 2004.

[16] Ryan Leigland and Axel W. Krings. A formalization of digital forensics. *International Journal of Digital Evidence*, 3, Issue 2, 2004.

[17] Matthew Meyers and Marc Rogers. Computer forensics: The need for standardization and certification. *International Journal of Digital Evidence*, 3, Issue 2, 2004.

[18] Lin Padgham and Michael Winikoff. *Developing Intelligent Agent Systems: A Practical Guide*. John Wiley and Sons, 1. edition, 2004.

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

[20] Yongping Tang and Thomas E. Daniels. A simple framework for distributed forensics. In *Second International Workshop on Security in Distributed Computing Systems*, pages 163–169, 2005.

[21] Chris van Aart. Ontology Bean Generator. `http://acklin.nl/page.php?id=34`, accessed March 2006.

[22] Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu. Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD. Cryptology ePrint Archive, Report 2004/199, 2004.

[23] Ren Wei. A framework of distributed agent-based network forensics system. Presented at the Digital Forensic Research Workshop 2004, August 2004.

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

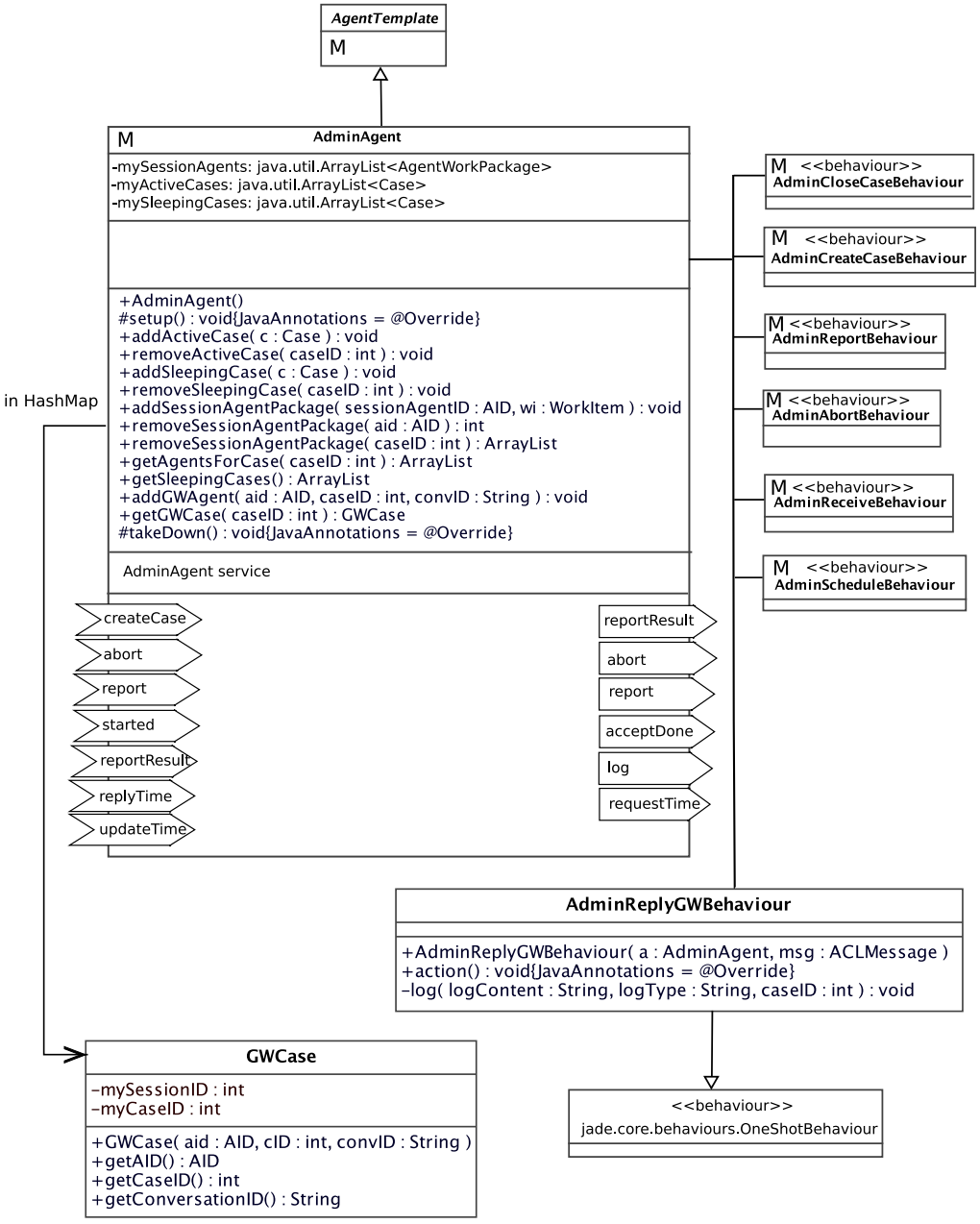


Figure B.1: The AUML class diagram of AdminAgent and its behaviours.

B.1.2 SessionAgent

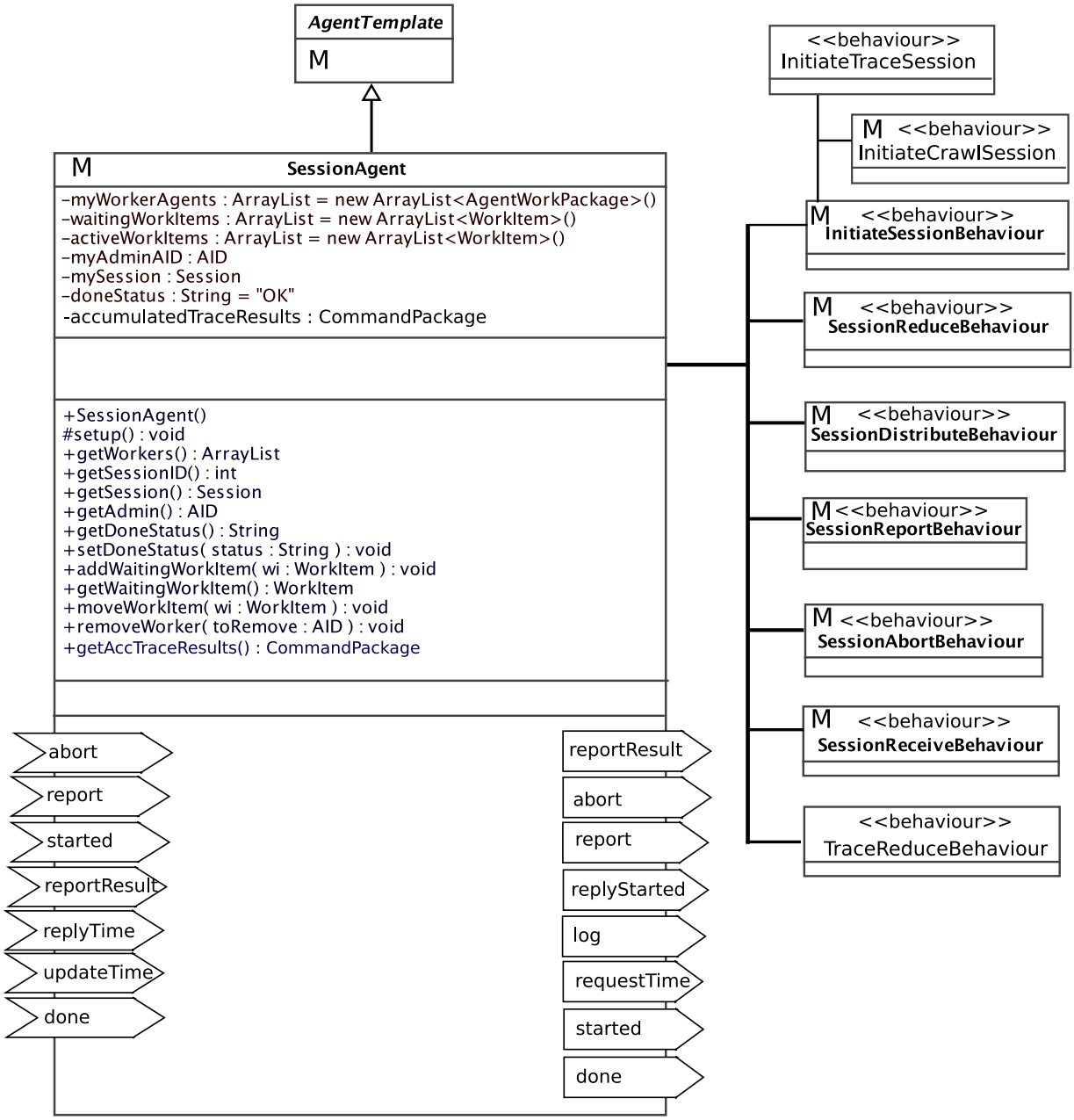


Figure B.2: The AUML class diagram of SessionAgent.

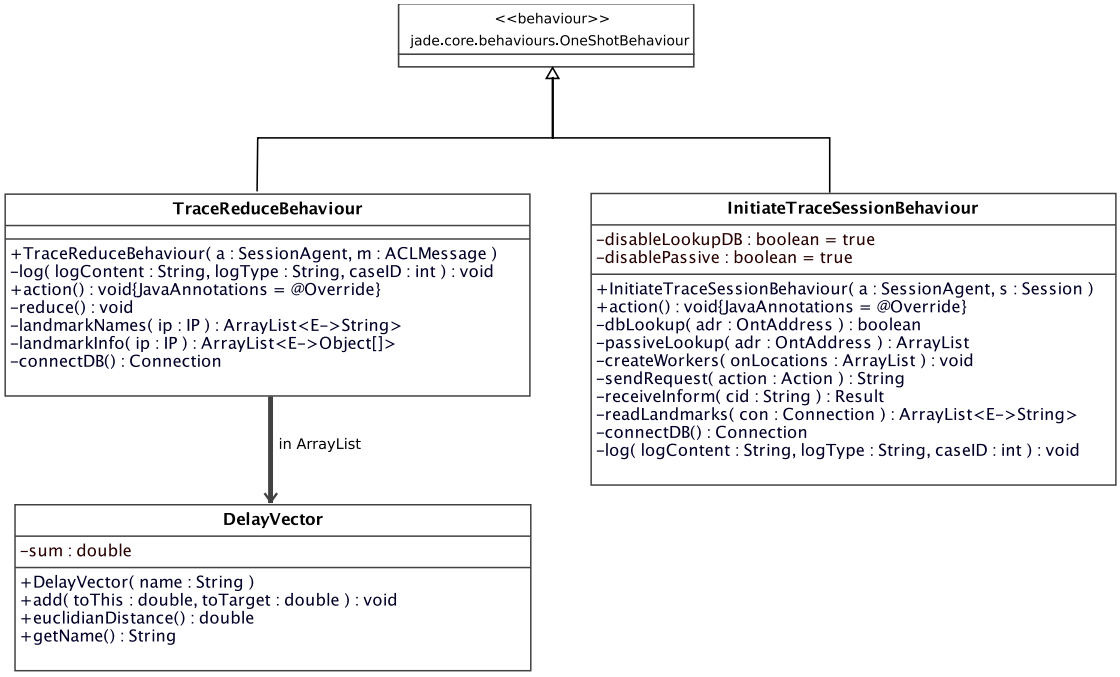


Figure B.3: The AUML class diagram of SessionAgent's behaviours.

B.1.3 WorkerAgent

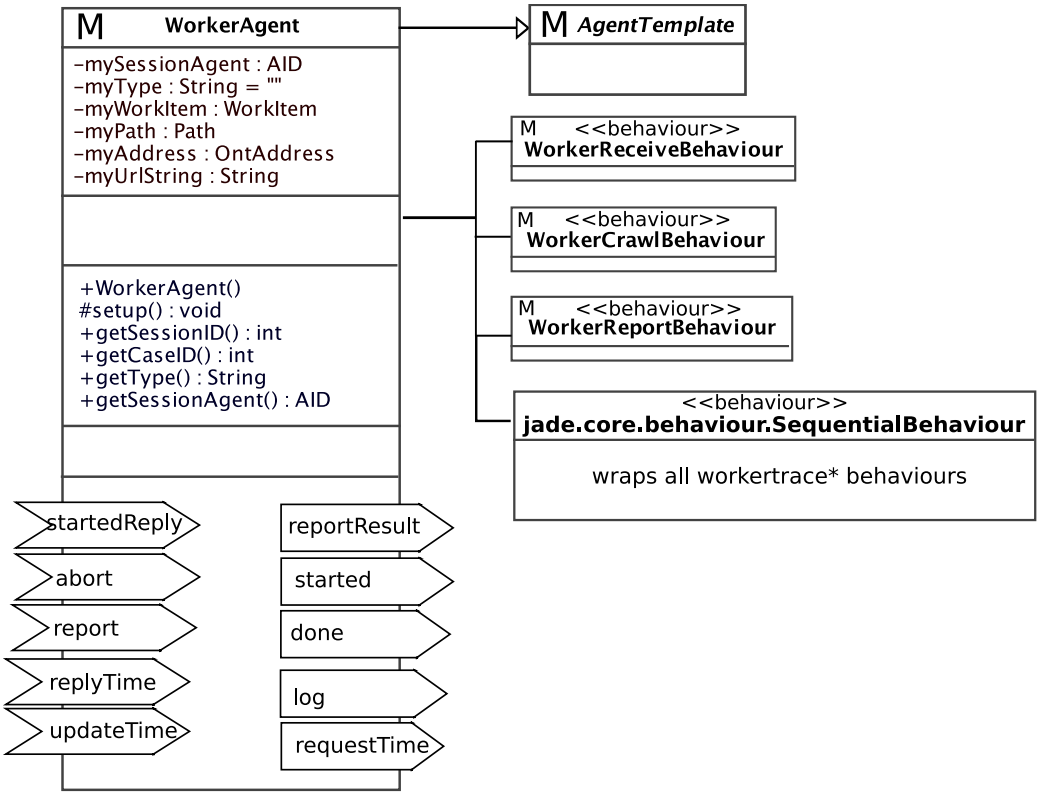


Figure B.4: The AUML class diagram of WorkerAgent.

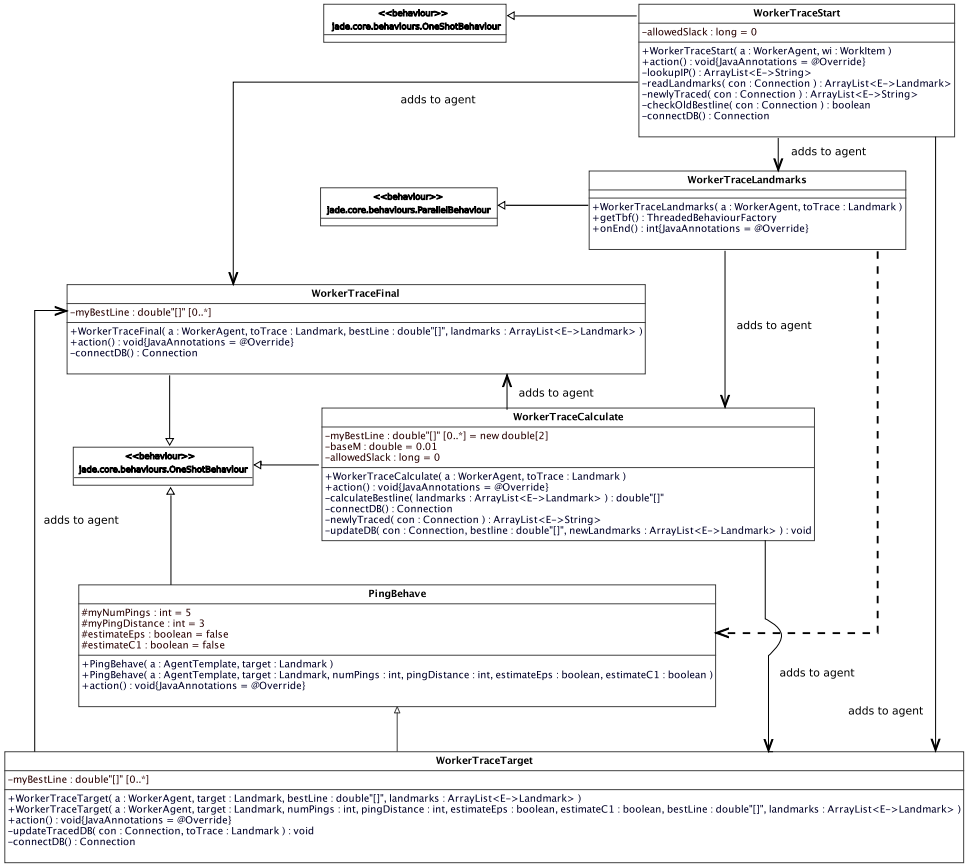


Figure B.5: The AUML class diagram of WorkerAgent's behaviours.

B.1.4 GWAgent

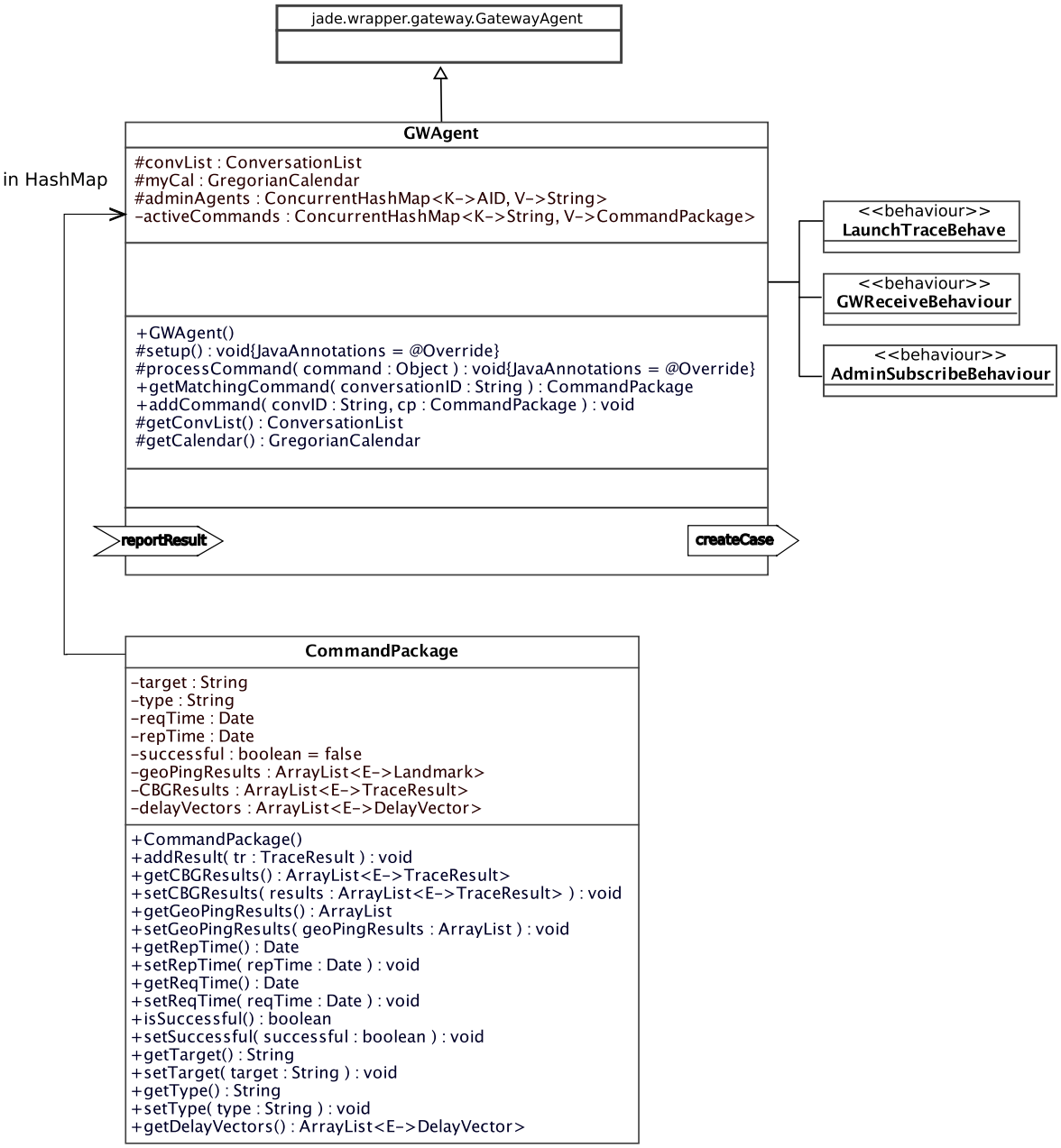


Figure B.6: The AUML class diagram of GWAgent.

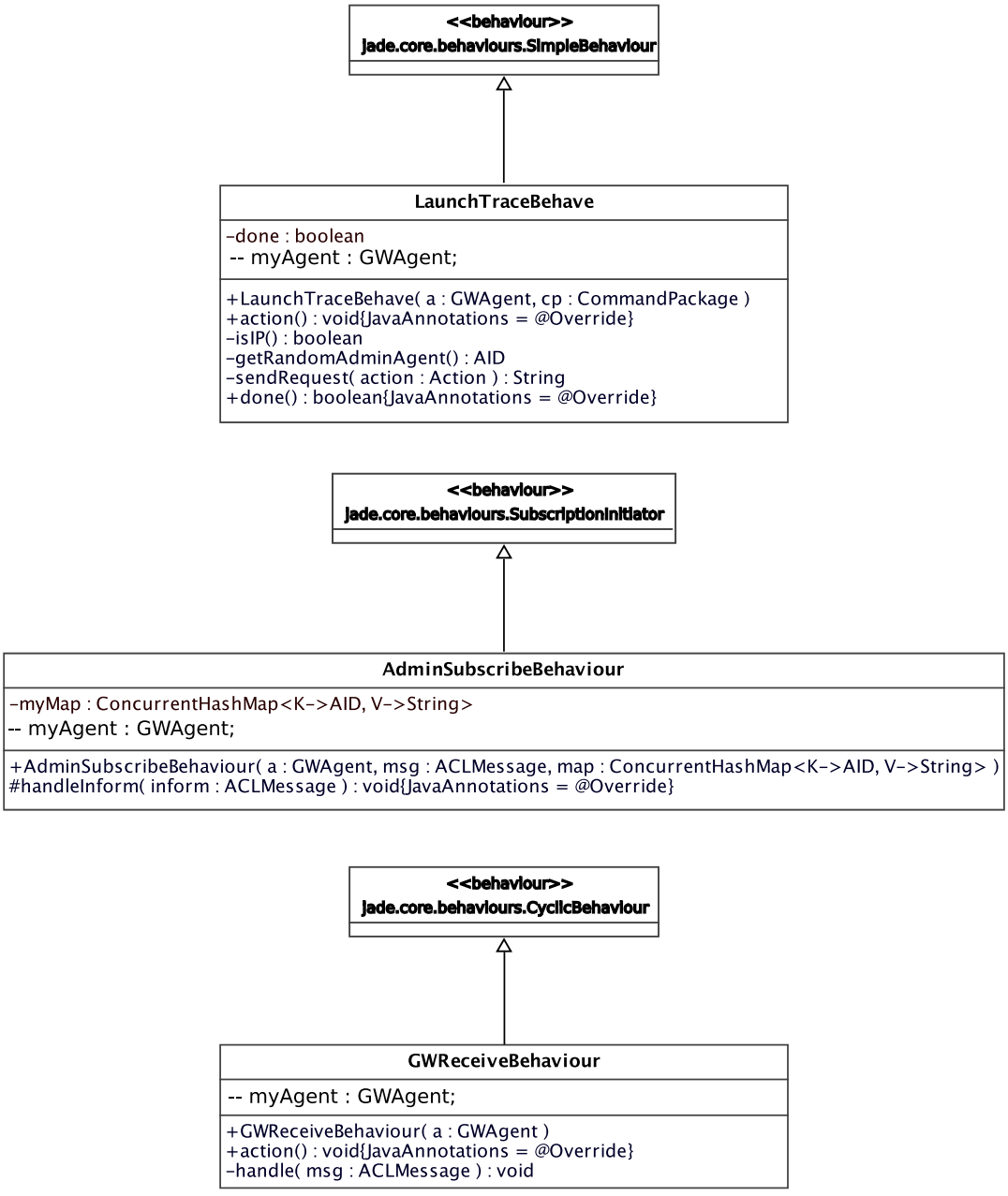


Figure B.7: The AUML class diagram of GWAgent's behaviours.

B.2 Servlet UML Diagrams

B.2.1 Servlet Core Classes

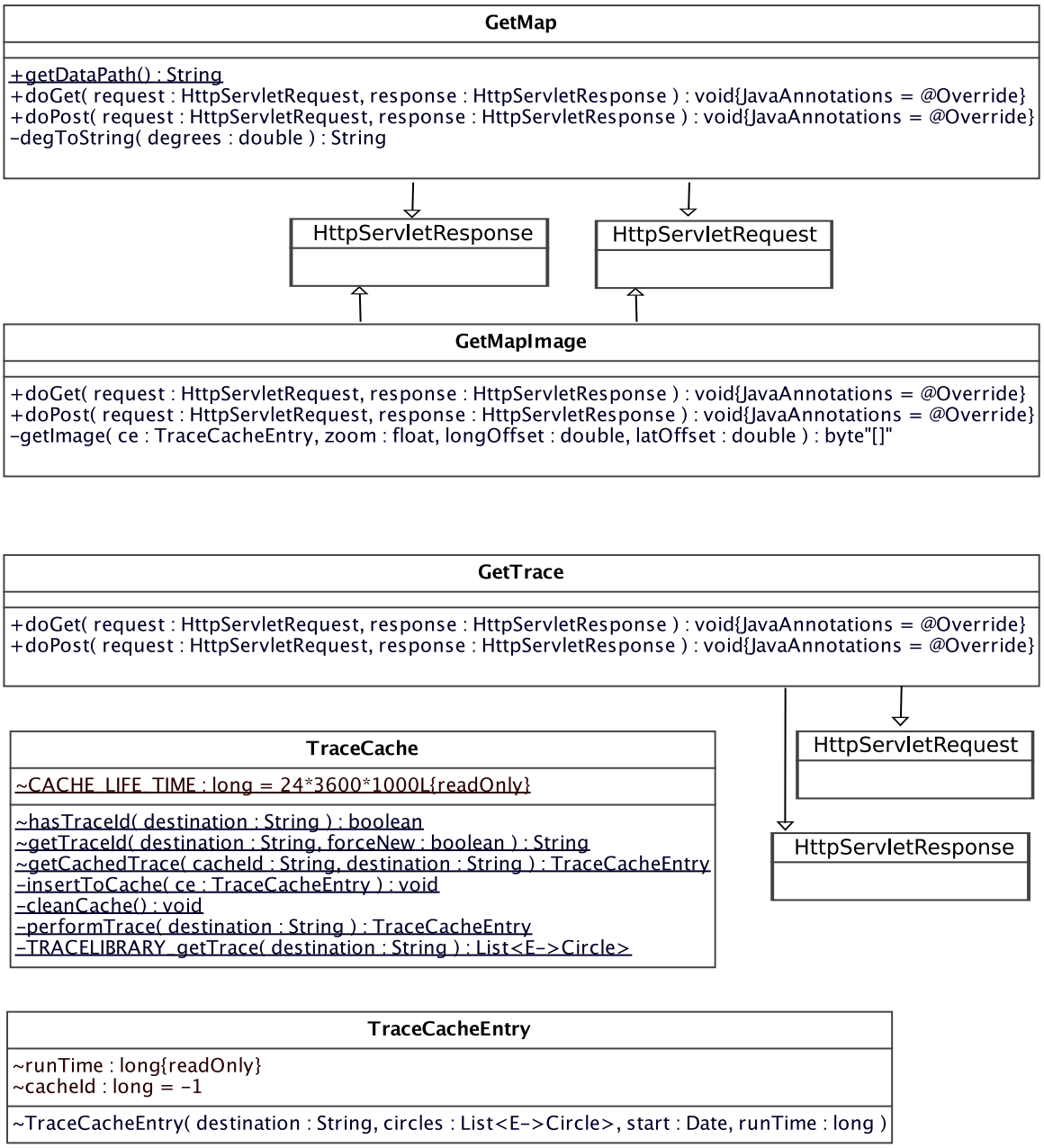


Figure B.8: The UML diagram of the core servlet classes.

B.2.2 Servlet Alpha Classes

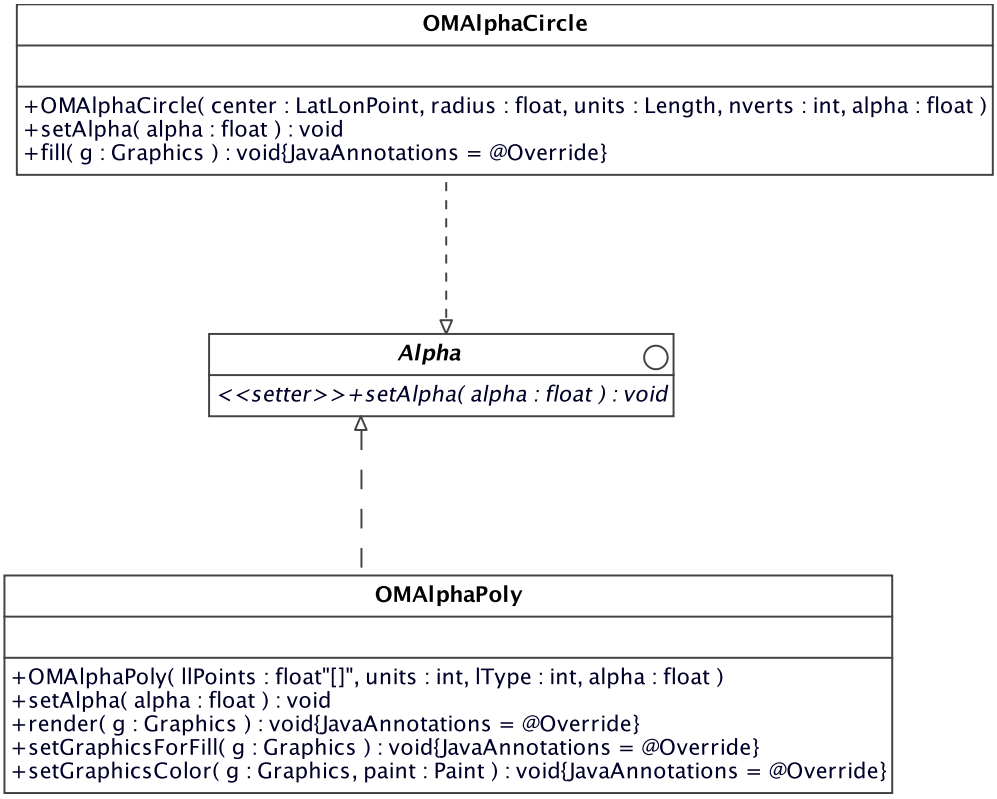


Figure B.9: The UML diagram of the servlet alpha classes.

B.2.3 Servlet Circle Classes

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

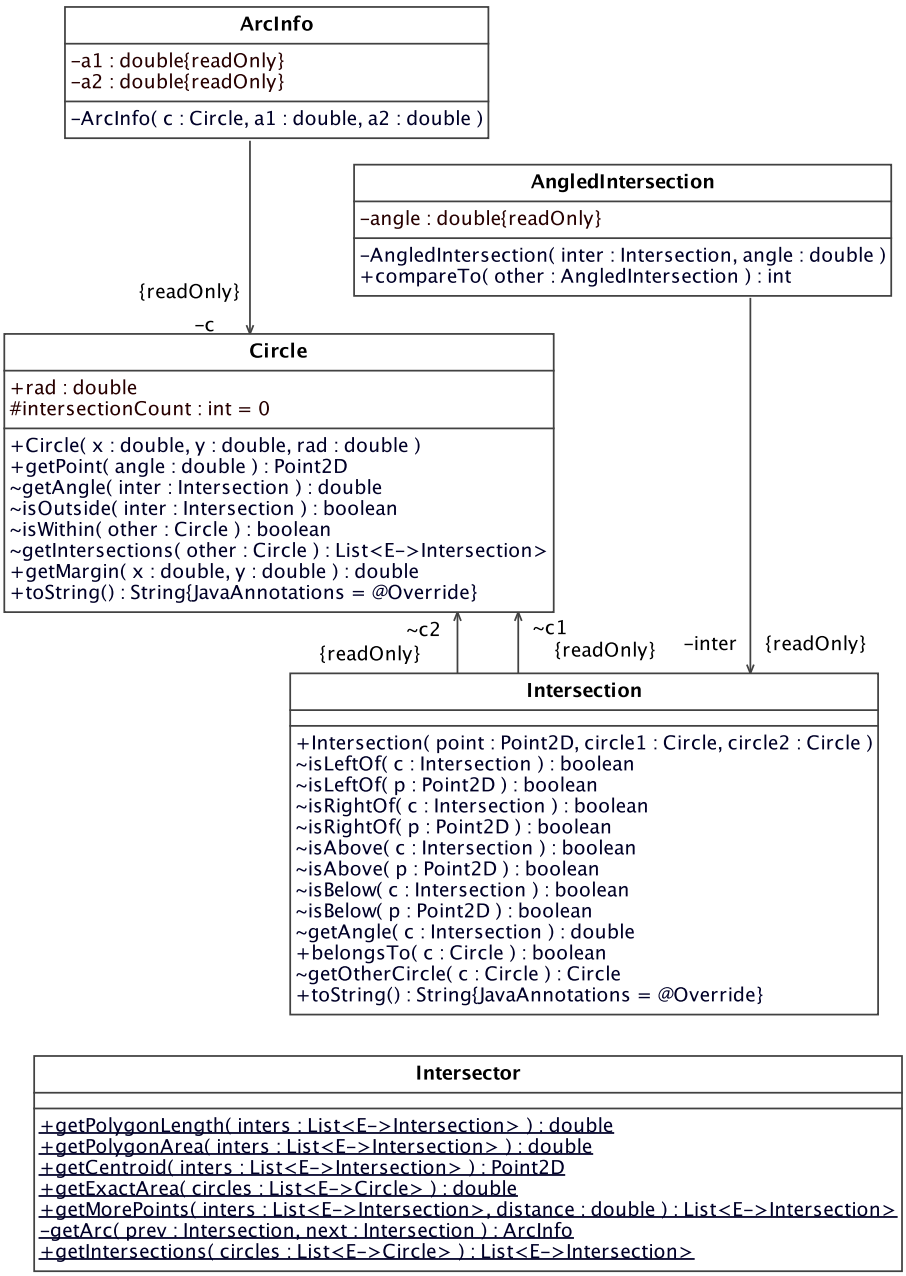


Figure B.10: The UML diagram of the servlet circle classes.

B.3 Database

B.3.1 UML Class Diagram

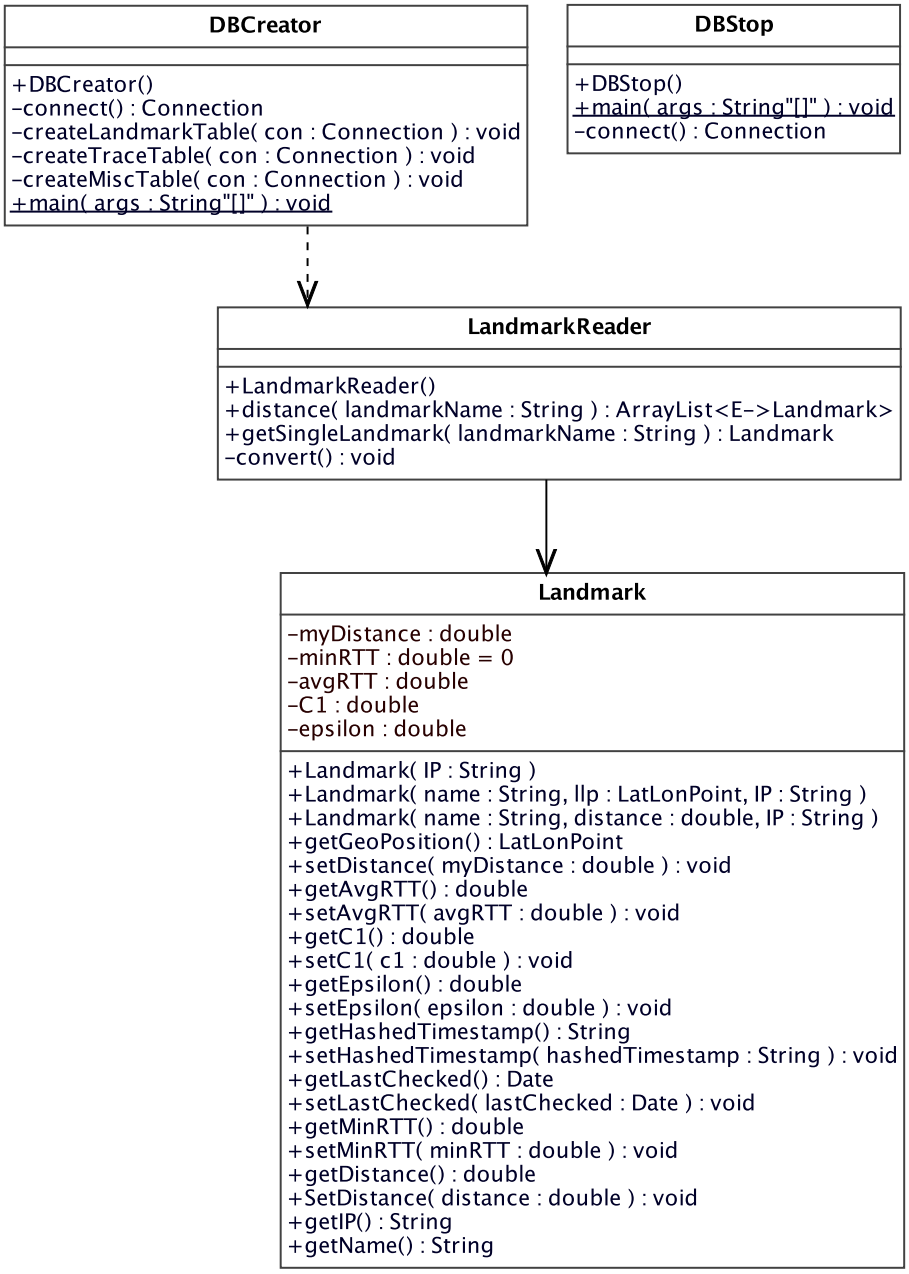


Figure B.11: The UML class diagram of database related classes.

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

```
1  package kripos.geo;
2
3  import jade.content.Concept;
4  import jade.content.ContentElement;
5  import jade.content.lang.Codec.CodecException;
6  import jade.content.onto.OntologyException;
7  import jade.content.onto.UngroundedException;
8  import jade.content.onto.basic.Action;
9  import jade.core.AID;
10 import jade.core.behaviours.OneShotBehaviour;
11 import jade.lang.acl.ACLMessage;
12 import kripos.ontology.LogRec;
13 import kripos.ontology.OntDate;
14 import kripos.ontology.TraceResultList;
15
16 /**
17  *
18  *
19  * @author oysteine
20  * @version 1.0
21  */
22 public class AdminReplyGWBehaviour extends OneShotBehaviour {
```

```
23    private static final long serialVersionUID =
          4025250051872250273L;
24    private AdminAgent myAdminAgent;
25    private ACLMessage myMsg;
26    private TraceResultList myTrl;
27
28    /**
29     * @param a
30     */
31    public AdminReplyGWBehaviour(AdminAgent a, ACLMessage msg) {
32      super(a);
33      myAdminAgent = a;
34      myMsg = msg;
35    }
36
37    /**
38     * Closes the case, logs appropriate information.
39     *
40     * @see jade.core.behaviours.Behaviour#action()
41     */
42    @Override
43    public void action() {
44      try {
45        AID aid = myMsg.getSender();
46        ContentElement ce = myAdminAgent.getContentManager().
              extractContent(myMsg);
47        Concept action = ((Action)ce).getAction();
48        myTrl = (TraceResultList)action;
49        String status = myTrl.getDoneStatus();
50
51        if(status.equalsIgnoreCase("OK")){
52
53          int cID = myAdminAgent.removeSessionAgentPackage(aid);
54          myAdminAgent.removeActiveCase(cID);
55          ACLMessage reply = myMsg.createReply();
56          reply.setPerformative(ACLMessage.INFORM);
57          myAgent.send(reply);
58
59          /* Logs that the Case has been closed*/
60          log("Case␣closed.␣SessionAgent␣terminated.␣Case␣end␣
                status:␣"+
61              myTrl.getDoneStatus(), "Type:Session", cID);
62          //send die to SessionAgent
63
64          //send data to GWAgent
65          GWCase gwCase = myAdminAgent.getGWCase(cID);
66          AID gwAID = gwCase.getAID();
67          String convID = gwCase.getConversationID();
68
69          //use myMsg for sending to GWAgent, already has correct
                content, only set envelopeinfo
70          myMsg.clearAllReceiver();
71          myMsg.clearAllReplyTo();
72          myMsg.addReceiver(gwAID);
```

```
73          myMsg.setConversationId(convID);
74          myMsg.setSender(myAdminAgent.getAID());
75          myAgent.send(myMsg);
76        }
77        else if(status.equalsIgnoreCase("Fail")){
78          int cID = myAdminAgent.removeSessionAgentPackage(aid);
79          myAdminAgent.removeActiveCase(cID);
80
81          //FIXME logging
82          /*
83                      log("Case closed. SessionAgent terminated.
                            Case end status: "+
84                            myTrl.getDoneStatus() + "\n" + "Case
                                FAILED due to: "
85                            + myTrl.getHasReason(), "Type:Session
                                ", cID);
86           */
87
88          //send data to GWAgent
89          AID gwAID = myAdminAgent.getGWCase(cID).getAID();
90          String convID = myAdminAgent.getGWCase(cID).
                getConversationID();
91
92          //use myMsg for sending to GWAgent, already has correct
                 content, only set envelopeinfo
93          myMsg.clearAllReceiver();
94          myMsg.clearAllReplyTo();
95          myMsg.addReceiver(gwAID);
96          myMsg.setConversationId(convID);
97          myMsg.setSender(myAdminAgent.getAID());
98          myAgent.send(myMsg);
99        }
100       else{
101         ACLMessage reply = myMsg.createReply();
102         reply.setPerformative(ACLMessage.NOT_UNDERSTOOD);
103         myAgent.send(reply);
104       }
105
106     } catch (UngroundedException e) {
107       e.printStackTrace();
108     } catch (CodecException e) {
109       e.printStackTrace();
110     } catch (OntologyException e) {
111       e.printStackTrace();
112     }
113   }//action()
114
115   /**
116    * Utility method for constructing LogRecs.
117    *
118    * @param logContent The content of the LogRec
119    * @param logType The Type of the LogRec
120    * @param caseID The CaseID of the LogRec
121    */
```

```
122     private void log(String logContent, String logType, int
            caseID){
123       LogRec logr = new LogRec();
124       logr.setHasCaseID(caseID);
125       //logr.setHasSessionID(myCase.getCaseSessions().
              getSessionID());
126       logr.setLogType(logType);
127       OntDate od = new OntDate();
128       od.setTime(myAdminAgent.getCalendar().getTimeInMillis());
129       logr.setHasDate(od);
130       logr.setLogLevel("INFO");
131       logr.setLogContent(logContent);
132       myAdminAgent.getLogger().log(logr);
133     }
134
135   }//class
```

C.1.2 GWCase

```
1  package kripos.geo;
2
3  import jade.core.AID;
4
5  /**
6   * Utility class that bundles AID and Case information.
7   * For use by AdminAgents to keep track of which GWAgent
        requested which Case
8   *
9   * @author oysteine
10  * @version 1.0
11  */
12 public class GWCase {
13   private AID myAID;
14   private int mySessionID;
15   private int myCaseID;
16   private String myConversationID;
17
18   /**
19    * Constructs an instance of GWCase
20    *
21    * @param aid the AID of an agent
22    * @param cID the case ID of a Case
23    */
24   public GWCase(AID aid, int cID, String convID) {
25     super();
26     myAID = aid;
27     myCaseID = cID;
28     myConversationID = convID;
29   }
30
31   /**
32    * @return the myAID
33    */
34   public AID getAID() {
35     return myAID;
36   }
37
38   /**
39    * @return the myCaseID
40    */
41   public int getCaseID() {
42     return myCaseID;
43   }
44
45   /**
46    *
47    * @return the myConversationID
48    */
49   public String getConversationID(){
50     return myConversationID;
51   }
```

```
52
53 }//class
```

C.2 SessionAgent Classes

C.2.1 InitiateTraceSessionBehaviour

```
1  package kripos.geo;
2
3  import java.sql.Connection;
4  import java.sql.DriverManager;
5  import java.sql.ResultSet;
6  import java.sql.SQLException;
7  import java.sql.Statement;
8  import java.util.ArrayList;
9  import java.util.Random;
10 import jade.content.ContentElement;
11 import jade.content.lang.Codec.CodecException;
12 import jade.content.lang.sl.SLCodec;
13 import jade.content.onto.OntologyException;
14 import jade.content.onto.UngroundedException;
15 import jade.content.onto.basic.Action;
16 import jade.content.onto.basic.Result;
17 import jade.core.AID;
18 import jade.core.ContainerID;
19 import jade.core.Location;
20 import jade.core.behaviours.OneShotBehaviour;
21 import jade.domain.JADEAgentManagement.CreateAgent;
22 import jade.domain.JADEAgentManagement.
       QueryPlatformLocationsAction;
23 import jade.domain.JADEAgentManagement.WhereIsAgentAction;
24 import jade.domain.mobility.MobilityOntology;
25 import jade.lang.acl.ACLMessage;
26 import jade.lang.acl.MessageTemplate;
27 import kripos.gateway.CommandPackage;
28 import kripos.ontology.*;
29
30 /**
31  * Behaviour that initiates a trace
32  *
33  * @author oysteine
34  * @version 1.0
35  */
36 public class InitiateTraceSessionBehaviour extends
       OneShotBehaviour {
37   private static final long serialVersionUID =
         -5048721800518980650L;
38   //private ArrayList<Location> locations = new ArrayList<
       Location>();
39   private SessionAgent mySessionAgent;
40   private Session mySession;
41   private boolean disableLookupDB = true;
42   private boolean disablePassive = true;
43
44   /**
45    * Creates an instance of this behaviour.
```

```
46      *
47      * @param a The agent this instance belongs to.
48      */
49     public InitiateTraceSessionBehaviour(SessionAgent a, Session
            s) {
50       super(a);
51       mySessionAgent = a;
52       mySession = s;
53     }
54
55     /**
56      * Does initial work based on the type of trace session.
57      *    *
58      * @see jade.core.behaviours.OneShotBehaviour#action()
59      */
60     @Override
61     public void action(){
62       OntAddress toTrace = mySession.getHasBaseAddress();
63       String params = mySession.getSessionParameters();
64       //parse and check if dblookup
65       //parse and check if passive
66       //parse and check random-period for when to ping at a
67
68       String sessionType = mySession.getSessionType();
69
70       //fill commandpackage in owning SessionAgent
71       CommandPackage atr = mySessionAgent.getAccTraceResults();
72       atr.setReqTime(mySession.getSessionStarted());
73       if(toTrace instanceof IP) {
74         atr.setTarget(((IP)toTrace).getHasName());
75       }
76       else if(toTrace instanceof DNSname){
77         atr.setTarget(((DNSname)toTrace).getHasName());
78       }
79       atr.setType(sessionType);
80
81       //TODO SEND 3 PINGS TO CHECK IF REACHABLE AT ALL BEFORE
             WASTING RESOURCES
82
83       //TODO db lookup
84       if(!disableLookupDB){
85         dbLookup(toTrace);
86       }
87
88       if(!disablePassive){
89 //      //TODO must return list of locations to use for trace
90         ArrayList locations = passiveLookup(toTrace);
91         //TODO superlandmark from this container?
92         createWorkers(locations);
93       }
94       else {
95         //TODO superlandmark from this container?
96         createWorkers(null);
97       }
```

```
98
99
100 //  /* Logs the start of the Session*/ TODO log correct
        information
101 //  log("Session started. WorkerAgents created" + "\n" +
102 //  "SessionType: " + mySession.getSessionType() +
103 //  "\n" + "Target: "+ mySession.getHasProtocol().getHasPrefix
        () +
104 //  mySession.getHasBaseAddress()+mySession.getHasPath().
        toString(),
105 //  "Type:Session", mySession.getCaseID());
106
107   }
108
109   /**
110    * TODO
111    *
112    * @param adr
113    * @return
114    */
115   private boolean dbLookup(OntAddress adr){ //not return true?
116     boolean recent = false;
117     //les inn terskel fra properties?
118     //dboppslag
119
120     //if(){} break and return traceresult directly from db.
121     //else{
122     return recent;
123     //  }
124   }
125
126   /**
127    * TODO
128    *
129    * @param adr
130    * @return
131    */
132   private ArrayList passiveLookup(OntAddress adr){
133     ArrayList list = new ArrayList();
134     return list;
135   }
136
137   /**
138    * Creates the WorkerAgents that will perform the rest of
139    * the delay measurements of the trace operation.
140    *
141    * @param onLocations List of Locations to do measurements
           from.
142    * If null all available Locations are used.
143    */
144   private void createWorkers(ArrayList onLocations){
145     ArrayList<Location> locations = new ArrayList<Location>();
146     AID myAMS = myAgent.getAMS();
147
```

```
148       //query ams for platform available locations
149       QueryPlatformLocationsAction qpla = new
              QueryPlatformLocationsAction();
150       Action actAll = new Action(myAMS, qpla);
151       String convIDAll = sendRequest(actAll);
152
153       //receive locations
154       Result resAll = receiveInform(convIDAll);
155       jade.util.leap.Iterator it = resAll.getItems().iterator();
156
157       //uses all available locations
158       if(onLocations == null){
159         //due to unclear JadeGateway.init() documentation, need
              to filter out extra containers at mainhost
160         ArrayList<String> onlyInclude = readLandmarks(connectDB()
              );
161
162         while (it.hasNext()) {
163           Location loc = (Location)it.next();
164           for (int i=0;i<onlyInclude.size();i++){
165             if(loc.getName().equalsIgnoreCase(onlyInclude.get(i))
                  ){
166               locations.add(loc);
167               break;
168             }
169           }
170         }
171 //      //add own location
172 //      WhereIsAgentAction wia = new WhereIsAgentAction();
173 //      Action actSelf = new Action(myAMS, wia);
174 //      String convSelf = sendRequest(actSelf);
175
176 //      Result resSelf = receiveInform(convSelf);
177 //      ContainerID cid = (ContainerID)resSelf.getValue();
178
179 //      locations.add(cid);
180       }
181
182       //uses only supplied locations
183       else{
184         while (it.hasNext()) {
185           Location loc = (Location)it.next();
186           for(int i=0;i<onLocations.size();i++){ //should
                optimize by removing from onLocation if match
187             if(loc.getName().equalsIgnoreCase((String)onLocations
                    .get(i))){
188               locations.add(loc);
189             }
190           }
191         }
192       }
193
194       //actually create agents
195       for (int i=0; i < locations.size(); i++){
```

```
196         CreateAgent ca = new CreateAgent();
197         ca.setAgentName("Worker" + mySession.getSessionID() +"␣"+
                i);
198         ca.setClassName("kripos.geo.WorkerAgent");
199         ContainerID cid = (ContainerID)locations.get(i);
200         ca.setContainer(cid);
201         ca.addArguments(myAgent.getName());
202
203         WorkItem wi = new WorkItem();
204         wi.setCaseID(mySession.getCaseID());
205         wi.setSessionID(mySession.getSessionID());
206         wi.setHasBaseAddress(mySession.getHasBaseAddress());
207         wi.setSessionType(mySession.getSessionType());
208         ca.addArguments(wi);
209         Action a2 = new Action(myAMS, ca);
210         String convID2 = sendRequest(a2);
211
212         MessageTemplate mt = MessageTemplate.MatchConversationId(
                convID2);
213         ACLMessage resp = myAgent.blockingReceive(mt, 20000);
214
215         mySessionAgent.getConvList().deregisterConversation(resp.
                getConversationId());
216       }
217
218       locations.clear();
219     }
220
221     /**
222      * Utility method for sending ACLMessages
223      * <p>
224      * Registers the conversationID in the convList
225      * to avoid loosing any reply to the generic receivebehaviour
226      *
227      * @param action the action that is to be wrapped in an
             ACLMessage
228      * @return
229      */
230     private String sendRequest(Action action){
231       try {
232         ACLMessage qMsg = new ACLMessage(ACLMessage.REQUEST);
233         qMsg.setConversationId(myAgent.getName() + new Random().
                nextLong());
234         qMsg.setLanguage(new SLCodec(0).getName());
235
236         qMsg.setOntology(MobilityOntology.getInstance().getName()
                );
237         myAgent.getContentManager().fillContent(qMsg, action);
238         qMsg.addReceiver(action.getActor());
239         //register conversation with agent to get correct
                reception
240         mySessionAgent.getConvList().registerConversation(qMsg.
                getConversationId());
241         myAgent.send(qMsg);
```

```
242         return qMsg.getConversationId();
243       }
244       catch (CodecException e) {
245         e.printStackTrace();
246         return null;
247       }
248       catch (OntologyException e) {
249         e.printStackTrace();
250         return null;
251       }
252     }
253     /**
254      * Utility method for receiving ACLMessages with mathcing
            conversation IDs
255      *
256      * @param cid the conversation id to match
257      * @return the content of the ACLMessage
258      */
259     private Result receiveInform(String cid){
260       try {
261         MessageTemplate mt = MessageTemplate.MatchConversationId(
              cid);
262         ACLMessage resp = myAgent.blockingReceive(mt, 20000);
263         ContentElement ce = myAgent.getContentManager().
              extractContent(resp);
264         Result result = (Result) ce;
265         mySessionAgent.getConvList().deregisterConversation(cid);
266         return result;
267       }
268       catch (UngroundedException e) {
269         e.printStackTrace();
270         mySessionAgent.getConvList().deregisterConversation(cid);
271         return null;
272       }
273       catch (CodecException e) {
274         e.printStackTrace();
275         mySessionAgent.getConvList().deregisterConversation(cid);
276         return null;
277       }
278       catch (OntologyException e) {
279         e.printStackTrace();
280         mySessionAgent.getConvList().deregisterConversation(cid);
281         return null;
282       }
283     }
284
285     /**
286      * Reads landmarks from the local database.
287      *
288      * @param con Database connection
289      * @return a list of landmarks
290      */
291     private ArrayList<String> readLandmarks(Connection con){
292       ArrayList<String> landmarkNames = new ArrayList<String>();
```

```
293
294       try {
295         Statement stmt = con.createStatement();
296         String query = "SELECT␣NAME␣FROM␣LANDMARKS";
297         ResultSet rs = stmt.executeQuery(query);
298         while(rs.next()){
299           String name = rs.getString("NAME");
300           landmarkNames.add(name);
301         }
302       }catch (SQLException e) {
303         // TODO: handle exception
304       }
305       return landmarkNames;
306     }
307
308     /**
309      * Connects to the local database and returns the Connection
             for further use.
310      *
311      * @return a connection to the database
312      */
313     private Connection connectDB(){//TODO path & password
314       Connection c = null;
315       try {
316         Class.forName("org.hsqldb.jdbcDriver");
317         c = DriverManager.getConnection("jdbc:hsqldb:hsql://
                localhost/xdb", "sa", "");
318       }catch (ClassNotFoundException e) {
319         //FIXME unable to find database classes. dosomething
320         e.printStackTrace();
321       }
322       catch (SQLException e) {
323         //FIXME unable to connect to database dosomething
324         e.printStackTrace();
325       }
326       return c;
327     }
328
329     /**
330      * Utility method for constructing LogRecs  and logging them.
331      *
332      * @param logContent The content of the LogRec
333      * @param logType The Type of the LogRec
334      * @param caseID The CaseID of the LogRec
335      */
336     private void log(String logContent, String logType, int
            caseID){
337       LogRec logr = new LogRec();
338       logr.setHasCaseID(caseID);
339       logr.setHasSessionID(mySession.getSessionID());
340       logr.setLogType(logType);
341       OntDate od = new OntDate();
342       od.setTime(mySessionAgent.getCalendar().getTimeInMillis());
343       logr.setHasDate(od);
```

```
344       logr.setLogLevel("INFO");
345       logr.setLogContent(logContent);
346       mySessionAgent.getLogger().log(logr);
347     }
348 
349 }//class
```

C.2.2 TraceReduceBehaviour

```
1  package kripos.geo;
2
3  import java.sql.Connection;
4  import java.sql.DriverManager;
5  import java.sql.ResultSet;
6  import java.sql.SQLException;
7  import java.sql.Statement;
8  import java.util.ArrayList;
9  import java.util.Iterator;
10
11 import jade.content.ContentElement;
12 import jade.content.lang.Codec.CodecException;
13 import jade.content.lang.sl.SLCodec;
14 import jade.content.onto.OntologyException;
15 import jade.content.onto.UngroundedException;
16 import jade.content.onto.basic.Action;
17 import jade.core.AID;
18 import jade.core.behaviours.OneShotBehaviour;
19 import jade.lang.acl.ACLMessage;
20 import jade.lang.acl.MessageTemplate;
21 import kripos.gateway.CommandPackage;
22 import kripos.ontology.*;
23
24 /**
25  * Used by SessionAgent when a WorkerAgent is finished and
        reports in
26  *
27  * @author oysteine
28  * @version 1.0
29  */
30 public class TraceReduceBehaviour extends OneShotBehaviour{
31   private static final long serialVersionUID =
         -7988728778910969955L;
32   private SessionAgent mySessionAgent;
33   private ACLMessage myMsg;
34   private AID myWorkerAID;
35   private IP myTarget;
36   private String myType = null;
37   private String myWorkerDoneStatus = "OK";
38
39   /**
40    * Creates an instance of this Behaviour.
41    *
42    * @param a the SessionAgent that owns this instance
43    * @param m the ACLMessage this behaviour will work on
44    */
45   public TraceReduceBehaviour(SessionAgent a, ACLMessage m) {
46     super(a);
47     mySessionAgent = a;
48     myMsg = m;
49   }
50
```

```
51    /**
52     * Utility method for constructing LogRecs and logging them.
53     *
54     * @param logContent The content of the LogRec
55     * @param logType The Type of the LogRec
56     * @param caseID The CaseID of the LogRec
57     */
58    private void log(String logContent, String logType, int
          caseID){
59      LogRec logr = new LogRec();
60      logr.setHasCaseID(caseID);
61      logr.setHasSessionID(mySessionAgent.getSession().
            getSessionID());
62      logr.setLogType(logType);
63      OntDate od = new OntDate();
64      od.setTime(mySessionAgent.getCalendar().getTimeInMillis());
65      logr.setHasDate(od);
66      logr.setLogLevel("INFO");
67      logr.setLogContent(logContent);
68      mySessionAgent.getLogger().log(logr);
69    }
70
71    /* (non-Javadoc)
72     * @see jade.core.behaviours.Behaviour#action()
73     */
74    @Override
75    public void action() {
76      //TODO update db?
77      try {
78        myWorkerAID = myMsg.getSender();
79        ContentElement ce = mySessionAgent.getContentManager().
              extractContent(myMsg);
80        TraceResultList trl = (TraceResultList)((Action)ce).
              getAction();
81        ArrayList<TraceResult> traceResults = new ArrayList<
              TraceResult>();
82        Iterator it = trl.getAllTraceResult();
83        while(it.hasNext()){
84          traceResults.add((TraceResult)it.next());
85        }
86        myTarget = (IP)(traceResults.get(0)).getAddressToBeTraced
              ();
87
88        if(traceResults.size()<2){//TODO check type instead!!
89          //TODO check for error
90          myType = "TRACE-CBG";
91          trl.getDoneStatus(); //TODO set in total and discard
92          (mySessionAgent.getAccTraceResults()).addResult(
                traceResults.get(0));
93        }
94        else {
95          myType = "TRACE-GeoPing";
96          ArrayList<DelayVector> dVectors = mySessionAgent.
                getAccTraceResults().getDelayVectors();
```

```
97          if(dVectors.isEmpty()){
98            ArrayList<String> names = landmarkNames(myTarget);
99            for(int i=0;i<names.size();i++){
100             DelayVector dv = new DelayVector(names.get(i));
101             dVectors.add(dv);
102           }
103         }
104         for(int i=0;i<dVectors.size();i++){
105           for(int j=1;j<traceResults.size();j++){//skip the
                TraceResult to target
106             if(dVectors.get(i).getName().equalsIgnoreCase(((
                  DNSname)(traceResults.get(j).
                  getAddressToBeTraced())).getHasName())){
107               dVectors.get(i).add(traceResults.remove(j).
                    getTraceResultData(), traceResults.get(0).
                    getTraceResultData());
108             }
109           }
110         }
111       }
112
113       //TODO myWorkerDoneStatus = done.getDoneStatus(); //fix
            tilsvarende i trace
114       //TODO mySessionAgent.setDoneStatus(myWorkerDoneStatus);
115
116       if(myWorkerDoneStatus.equalsIgnoreCase("OK")){
117         reduce();
118
119         log("WorkerAgent:" + myWorkerAID +
120             "finished␣successfully,␣terminated", "Type:Session"
                ,
121             mySessionAgent.getSession().getCaseID());
122       }
123       else if(myWorkerDoneStatus.equalsIgnoreCase("Fail")){
124         //Should do something more. Why failed? Create new
              worker?
125         reduce();
126
127         log("WorkerAgent:" + myWorkerAID + "failed␣due␣to" +//
              FIXME done.getHasReason() +
128             ",␣terminated", "Type:Session",
129             mySessionAgent.getSession().getCaseID());
130       }
131       else{
132         ACLMessage reply = myMsg.createReply();
133         reply.setPerformative(ACLMessage.NOT_UNDERSTOOD);
134         myAgent.send(reply);
135       }
136
137     } catch (UngroundedException e) {
138       e.printStackTrace();
139     } catch (CodecException e) {
140       e.printStackTrace();
141     } catch (OntologyException e) {
```

```
142         e.printStackTrace();
143       }
144     }//action()
145
146     /**
147      * Cleans up after a WorkerAgent is Done, and optionally
            terminates the entire Session.
148      * <p>
149      * Removes the workerAgent that sent the TraceList from
            myWorkerAgents.
150      * Sets the doneStatus of the sessionAgent to the doneStatus
            in the incoming Done.
151      * Checks if there are any more WorkerAgents, if not
            terminate the session and kill
152      * the owning SessionAgent after informing its AdminAgent.
153      */
154     private void reduce(){
155       mySessionAgent.removeWorker(myWorkerAID);
156       //TODO mySessionAgent.setDoneStatus(myWorkerDoneStatus);
157
158       if (mySessionAgent.getWorkers().isEmpty()){
159         TraceResultList trl = new TraceResultList();
160         CommandPackage cp = mySessionAgent.getAccTraceResults();
161         if(myType.equalsIgnoreCase("Trace-CBG")){
162           ArrayList<TraceResult> results = cp.getCBGResults();
163           for(int i=0;i<results.size();i++){
164             trl.addTraceResult(results.get(i));
165           }
166
167         }
168         else if(myType.equalsIgnoreCase("Trace-GeoPing")){
169           ArrayList<DelayVector> dList = cp.getDelayVectors();
170           ArrayList<Object[]> lInfo = landmarkInfo(myTarget);
171
172           double one = 99999999;//arbitrary high values
173           double two = 99999999;
174           double three = 99999999;
175           double four = 99999999;
176           TraceResult tr1 = new TraceResult();
177           TraceResult tr2 = new TraceResult();
178           TraceResult tr3 = new TraceResult();
179           TraceResult tr4 = new TraceResult();
180
181           for (int i=0;i<dList.size();i++){
182             Double temp = dList.get(i).euclidianDistance();
183             if(temp<one && temp>0){
184               one = temp;
185               for(int j=0;j<lInfo.size();j++){
186                 if(dList.get(i).getName().equalsIgnoreCase((
                      String)lInfo.get(j)[0])){
187                   GeoLocation geo = new GeoLocation();
188                   geo.setLocationName((String)lInfo.get(j)[0]);
189                   geo.setLocationLatitude(((Double)lInfo.get(j)
                        [1]).floatValue());
```

```
190                 geo.setLocationLongitude(((Double)lInfo.get(j)
                        [2]).floatValue());
191                 tr1.setHasGeoLocation(geo);
192               }
193             }
194             tr1.setTraceResultData(temp.floatValue());
195           }
196           else if(temp<two && temp>0){
197             two = temp;
198             for(int j=0;j<lInfo.size();j++){
199               if(dList.get(i).getName().equalsIgnoreCase((
                      String)lInfo.get(j)[0])){
200                 GeoLocation geo = new GeoLocation();
201                 geo.setLocationName((String)lInfo.get(j)[0]);
202                 geo.setLocationLatitude(((Double)lInfo.get(j)
                        [1]).floatValue());
203                 geo.setLocationLongitude(((Double)lInfo.get(j)
                        [2]).floatValue());
204                 tr2.setHasGeoLocation(geo);
205               }
206             }
207             tr2.setTraceResultData(temp.floatValue());
208           }
209
210           else if(temp<three && temp>0){
211             three = temp;
212             for(int j=0;j<lInfo.size();j++){
213               if(dList.get(i).getName().equalsIgnoreCase((
                      String)lInfo.get(j)[0])){
214                 GeoLocation geo = new GeoLocation();
215                 geo.setLocationName((String)lInfo.get(j)[0]);
216                 geo.setLocationLatitude(((Double)lInfo.get(j)
                        [1]).floatValue());
217                 geo.setLocationLongitude(((Double)lInfo.get(j)
                        [2]).floatValue());
218                 tr3.setHasGeoLocation(geo);
219               }
220             }
221             tr3.setTraceResultData(temp.floatValue());
222           }
223
224           else if(temp<four && temp>0){
225             four = temp;
226             for(int j=0;j<lInfo.size();j++){
227               if(dList.get(i).getName().equalsIgnoreCase((
                      String)lInfo.get(j)[0])){
228                 GeoLocation geo = new GeoLocation();
229                 geo.setLocationName((String)lInfo.get(j)[0]);
230                 geo.setLocationLatitude(((Double)lInfo.get(j)
                        [1]).floatValue());
231                 geo.setLocationLongitude(((Double)lInfo.get(j)
                        [2]).floatValue());
232                 tr4.setHasGeoLocation(geo);
233               }
```

```
234             }
235             tr4.setTraceResultData(temp.floatValue());
236           }
237
238         }
239         trl.addTraceResult(tr1);
240         trl.addTraceResult(tr2);
241         trl.addTraceResult(tr3);
242         trl.addTraceResult(tr4);
243       }
244
245       trl.setDoneStatus(mySessionAgent.getDoneStatus());
246       ACLMessage isDone = new ACLMessage(ACLMessage.INFORM);
247       isDone.addReceiver(mySessionAgent.getAdmin());
248       isDone.setLanguage(new SLCodec(0).getName());
249       isDone.setOntology(InternetInvestigationsOntology.
              getInstance().getName());
250       String convID = mySessionAgent.getConvList().
              registerConversation();
251       isDone.setConversationId(convID);
252       Action act = new Action(mySessionAgent.getAdmin(), trl);
253
254       try {
255         mySessionAgent.getContentManager().fillContent(isDone,
              act);
256         mySessionAgent.send(isDone);
257       } catch (CodecException e) {
258         e.printStackTrace();
259       } catch (OntologyException e) {
260         e.printStackTrace();
261       }
262
263       ACLMessage doneReply = mySessionAgent.blockingReceive(
              MessageTemplate.MatchConversationId(convID), 90000);
264       if (doneReply != null){
265         if (doneReply.getPerformative() == ACLMessage.INFORM);
266         /* Log that we have reported done and terminated after
              receiving reply*/
267         log(myAgent.getName() + "finished", "Type:Session",
268             mySessionAgent.getSession().getCaseID());
269
270         myAgent.doDelete();
271       }
272       else{
273         /* Log that we have reported done and terminated on
              timeout*/
274         log(myAgent.getName() + "finished" + "selfterminated",
275             "Type:Session", mySessionAgent.getSession().
                 getCaseID());
276         myAgent.doDelete();
277       }
278     }//outer if
279
280
```

```
281   }//reduce()
282
283   /**
284    * Get the names of landmarks from the Landmarks database,
285    * excluding any landmark matching the input IP adress.
286    *
287    * @param IP address to exclude
288    * @return the number of landmarks
289    */
290   private ArrayList<String> landmarkNames(IP ip){
291     Connection con = connectDB();
292     ArrayList<String> names = new ArrayList<String>();
293     try {
294       Statement stmt = con.createStatement();
295       String query = "SELECT␣NAME␣FROM␣LANDMARKS␣WHERE␣IPADR␣
              <>'"+ip.getHasName()+"'";
296       ResultSet rs = stmt.executeQuery(query);
297       while(rs.next()){
298         names.add(rs.getString(1));
299       }
300     }
301     catch(SQLException se){
302       se.printStackTrace();
303       return names;
304     }
305     return names;
306   }
307
308   /**
309    * Get the names and geographic locations of landmarks from
          the Landmarks database,
310    * excluding any landmark matching the input IP adress.
311    *
312    * @param IP address to exclude
313    * @return an array containing the name and latitude and
          longitude
314    */
315   private ArrayList<Object[]> landmarkInfo(IP ip){
316     Connection con = connectDB();
317     ArrayList<Object[]> info = new ArrayList<Object[]>();
318     try {
319       Statement stmt = con.createStatement();
320       String query = "SELECT␣NAME,␣LATITUDE,␣LONGITUDE␣FROM␣
              LANDMARKS␣WHERE␣IPADR␣<>'"+ip.getHasName()+"'";
321       ResultSet rs = stmt.executeQuery(query);
322       while(rs.next()){
323         Object[] element = new Object[3];
324         element[0] = rs.getString("NAME");
325         element[1] = new Double(rs.getDouble("LATITUDE"));
326         element[2] = new Double(rs.getDouble("LONGITUDE"));
327         info.add(element);
328       }
329     }
330     catch(SQLException se){
```

```
331         se.printStackTrace();
332         return info;
333       }
334       return info;
335     }
336 
337     /**
338      * Connects to the local database and returns the Connection
             for further use.
339      *
340      * @return a connection to the database
341      */
342     private Connection connectDB(){//TODO path & password
343       Connection c = null;
344       try {
345         Class.forName("org.hsqldb.jdbcDriver");
346         c = DriverManager.getConnection("jdbc:hsqldb:hsql://
                localhost/xdb", "sa", "");
347       }catch (ClassNotFoundException e) {
348         //FIXME unable to find database classes. dosomething
349         e.printStackTrace();
350       }
351       catch (SQLException e) {
352         //FIXME unable to connect to database dosomething
353         e.printStackTrace();
354       }
355       return c;
356     }
357 
358 }//class
```

C.2.3 DelayVector

```
1  package kripos.geo;
2
3  import java.util.ArrayList;
4
5  /**
6   * A delay vector for a given landmark.
7   *
8   * @author oysteine
9   * @version 1.0
10  */
11 public class DelayVector extends ArrayList {
12   private static final long serialVersionUID =
         7614366734429571963L;
13   private String myName;
14   private double sum;
15
16   /**
17    *
18    * @param name The name of the landmark represented by this
          instance
19    */
20   public DelayVector(String name) {
21     myName = name;
22   }
23
24   /**
25    * Adds a delay component to this DelayVector
26    *
27    * @param toThis the delay from calling landmark to this
          landmark
28    * @param toTarget the delay from calling landmark to the
          target
29    */
30   public void add(double toThis, double toTarget){
31     Double temp = new Double((toThis-toTarget)*(toThis-toTarget
          ));
32     this.add(temp);
33     sum = sum+temp;
34   }
35
36   /**
37    * Computes the Euclidian distance of this DelayVector
38    *
39    * @return the euclidian distance
40    */
41   public double euclidianDistance(){
42     if (sum>0){
43       return Math.sqrt(sum);
44     }
45     else {
46       return -1;
47     }
```

```
48    }
49
50    /**
51     *
52     * @return the name of the landmark represented by this
           DelayVector
53     */
54    public String getName(){
55      return myName;
56    }
57
58  }//class
```

C.3 WorkerAgent Classes

C.3.1 WorkerTraceStart

```
1  package kripos.geo;
2
3  import java.net.InetAddress;
4  import java.net.UnknownHostException;
5  import java.sql.Connection;
6  import java.sql.DriverManager;
7  import java.sql.ResultSet;
8  import java.sql.SQLException;
9  import java.sql.Statement;
10 import java.util.ArrayList;
11 import java.util.Date;
12 import com.bbn.openmap.LatLonPoint;
13 import jade.core.behaviours.OneShotBehaviour;
14 import jade.core.behaviours.SequentialBehaviour;
15 import kripos.ontology.DNSname;
16 import kripos.ontology.IP;
17 import kripos.ontology.OntAddress;
18 import kripos.ontology.WorkItem;
19
20 public class WorkerTraceStart extends OneShotBehaviour {
21   private static final long serialVersionUID =
         824433708805797669L;
22   private WorkerAgent myWorkerAgent;
23   private WorkItem myWorkItem;
24   private long allowedSlack = 0;//TODO set sane value
25
26   public WorkerTraceStart(WorkerAgent a, WorkItem wi) {
27     super(a);
28     myWorkerAgent = a;
29     myWorkItem = wi;
30   }
31
32   @Override
33   public void action() {
34     ArrayList<Landmark> landmarks = new ArrayList<Landmark>();
35     double[] bestLine = new double[2];
36     Landmark toTrace = null;
37     try {
38       toTrace = new Landmark(lookupIP().get(0));//TODO only
             uses first returned IP
39     } catch (UnknownHostException e1) {
40       e1.printStackTrace();
41     }
42
43     Connection con = connectDB();
44
45     ArrayList<Landmark> initialLandmarks = readLandmarks(con);
46     if(checkOldBestline(con)){
47       landmarks = initialLandmarks;
```

```
48        try {
49          Statement stmt = con.createStatement();
50          String query = "SELECT␣BESTLINE_M,␣BESTLINE_B␣FROM␣MISC
                ";
51          ResultSet rs = stmt.executeQuery(query);
52          while(rs.next()){
53            bestLine[0] = rs.getDouble("BESTLINE_M");
54            bestLine[1] = rs.getDouble("BESTLINE_B");
55          }
56
57          ArrayList<String> ips =  newlyTraced(con);
58          if (ips.get(ips.size()-1).equalsIgnoreCase("true")){
59            toTrace = new Landmark(ips.get(0));
60            try {
61              Statement st = con.createStatement();
62              String query2 = "SELECT␣NAME,␣MIN_RTT,␣C1,␣EPSILON,
                    ␣IPADR,␣CHECKED␣FROM␣TRACED␣WHERE␣IPADR␣=␣'"+ips
                    .get(0)+"'";
63              ResultSet rs2 = st.executeQuery(query2);
64              while(rs2.next()){
65                double minRtt = rs2.getDouble("MIN_RTT");
66                double c1 = rs2.getDouble("C1");
67                double epsilon = rs2.getDouble("EPSILON");
68                toTrace.setMinRTT(minRtt);
69                toTrace.setC1(c1);
70                toTrace.setEpsilon(epsilon);
71              }
72              ((SequentialBehaviour)root()).addSubBehaviour(
73                  new WorkerTraceFinal(myWorkerAgent, toTrace,
                        bestLine, landmarks));
74
75            } catch (SQLException e) {
76              e.printStackTrace();
77            }
78          }
79          else {
80            ((SequentialBehaviour)root()).addSubBehaviour(
81                new WorkerTraceTarget(myWorkerAgent, toTrace,
                      bestLine, landmarks));
82          }
83        }
84        catch(SQLException se){
85          //FIXME do something
86          se.printStackTrace();
87        }
88      }
89      else{
90        WorkerTraceLandmarks part2 = new WorkerTraceLandmarks(
              myWorkerAgent, toTrace);
91        myWorkerAgent.getPingList().setContent(initialLandmarks);
92        for (int i=0; i<initialLandmarks.size();i++){
93          PingBehave pb = new PingBehave(myWorkerAgent,
                initialLandmarks.get(i));
94          part2.addSubBehaviour(part2.getTbf().wrap(pb));
```

```
95          }
96
97          ((SequentialBehaviour)root()).addSubBehaviour(part2);
98        }
99
100     }//action
101
102     /**
103      * If the class of myTarget is DNSName we need to get the IP
            address.
104      * If the class of myTarget is IP no lookup is performed
            before returning the IP address.
105      *
106      * @return An ArrayList of IP addresses as strings
107      * @throws UnknownHostException
108      */
109     private ArrayList<String> lookupIP() throws
            UnknownHostException {
110       ArrayList<String> ipStrings = null;
111       OntAddress oa = myWorkItem.getHasBaseAddress();
112       if(oa instanceof DNSname) {
113         DNSname dns = (DNSname)(oa);
114         InetAddress[] ips;
115         ips = InetAddress.getAllByName(dns.getHasName());
116         ipStrings = new ArrayList<String>();
117         for(int i=0;i<ips.length;i++){
118           ipStrings.add(ips[i].getHostAddress());
119         }
120         return ipStrings;
121       }
122       else {
123         ipStrings = new ArrayList<String>();
124         ipStrings.add(((IP)myWorkItem.getHasBaseAddress()).
              getHasName());
125         return ipStrings;
126       }
127     }
128
129     /**
130      * Reads landmarks from the local database.
131      *
132      * @param con Database connection
133      * @return a list of landmarsk sorted by increasing distance
            from the owner of this instance
134      */
135     private ArrayList<Landmark> readLandmarks(Connection con){
136       ArrayList<Landmark> landmarks = new ArrayList<Landmark>();
137
138       try {
139         Statement stmt = con.createStatement();
140         String query = "SELECT␣*␣FROM␣LANDMARKS␣ORDER␣BY␣
              DISTANCE_KM␣ASC";
141         ResultSet rs = stmt.executeQuery(query);
142         while(rs.next()){
```

```
143           String name = rs.getString("NAME");
144           String ipAdr = rs.getString("IPADR");
145           double distance = rs.getDouble("DISTANCE_KM");
146           LatLonPoint llp = new LatLonPoint(rs.getDouble("
                  LATITUDE"), rs.getDouble("LONGITUDE"));
147
148           Landmark l = new Landmark(name, llp, ipAdr);
149           l.setDistance(distance);
150           landmarks.add(l);
151         }
152       }catch (SQLException e) {
153         // TODO: handle exception
154       }
155       return landmarks;
156     }
157
158
159     /**
160      * Checks if an IP address has currently been traced.
161      * The definition of current is given by allowedSlack.
162      *
163      * @param con The database connection to use
164      * @return true if this IP was recently traced
165      */
166     private ArrayList<String> newlyTraced(Connection con){
167       String newlyTraced = "false";
168       ArrayList<String> returnList = new ArrayList<String>();
169       try {
170         ArrayList<String> adrToCheck = lookupIP();
171         Statement stmt = con.createStatement(ResultSet.
                TYPE_SCROLL_INSENSITIVE,ResultSet.CONCUR_READ_ONLY);
172         for(int i=0;i<adrToCheck.size();i++){
173           String query = "SELECT␣NAME,␣IPADR,␣CHECKED␣FROM␣TRACED
                  ␣WHERE␣IPADR␣=␣'"+adrToCheck.get(i)+"'";
174           ResultSet rs = stmt.executeQuery(query);
175           if(!rs.next()){
176             returnList.add(adrToCheck.get(i));
177           }
178           rs.previous();
179           while (rs.next()) {
180             Date lastChecked  = (Date)rs.getTimestamp("CHECKED");
181             long toCheck;
182             if(rs.wasNull()){
183               toCheck = 0;
184             }
185             else{
186               toCheck = lastChecked.getTime();
187             }
188             long currentTime = myWorkerAgent.getCalendar().
                    getTimeInMillis();
189             long diff = currentTime - toCheck;
190             if(diff<allowedSlack){
191               returnList.add(rs.getString("IPADR"));
192               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
             host is too old.
216      *
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
231       } catch (SQLException e) {
232         e.printStackTrace();
233         return false;
234       }
235
236       long toCheck = lastChecked.getTime();
237       long currentTime = myWorkerAgent.getCalendar().
             getTimeInMillis();
238       long diff = currentTime - toCheck;
239
240       if(diff <= allowedSlack){
241         return true;
242       }
243       else{
```

```
244         return false;
245       }
246     }
247
248     /**
249      * Connects to the local database and returns the Connection
             for further use.
250      *
251      * @return a connection to the database
252      */
253     private Connection connectDB(){//TODO path & password
254       Connection c = null;
255       try {
256         Class.forName("org.hsqldb.jdbcDriver");
257         c = DriverManager.getConnection("jdbc:hsqldb:hsql://
                localhost/xdb", "sa", "");
258       }catch (ClassNotFoundException e) {
259         //FIXME unable to find database classes. dosomething
260         e.printStackTrace();
261       }
262       catch (SQLException e) {
263         //FIXME unable to connect to database dosomething
264         e.printStackTrace();
265       }
266       return c;
267     }
268
269 }//class
```

C.3.2 WorkerTraceLandmarks

```
1  package kripos.geo;
2
3  import jade.core.behaviours.ParallelBehaviour;
4  import jade.core.behaviours.SequentialBehaviour;
5  import jade.core.behaviours.ThreadedBehaviourFactory;
6
7  /**
8   * Wrapper for multiple PingBehaves running in dedicated
        threads
9   *
10  * @author oysteine
11  *
12  */
13 public class WorkerTraceLandmarks extends ParallelBehaviour {
14   private static final long serialVersionUID =
         -2880427229534140463L;
15   private ThreadedBehaviourFactory myTbf;
16   private Landmark myTarget;
17   private WorkerAgent myWorkerAgent;
18
19   public WorkerTraceLandmarks(WorkerAgent a, Landmark toTrace)
         {
20     super(a, ParallelBehaviour.WHEN_ALL);
21     myTbf = new ThreadedBehaviourFactory();
22     myTarget = toTrace;
23     myWorkerAgent = a;
24   }
25
26   /**
27    *
28    * @return the <code>ThreadedBehaviourFactory</code> of this
          instance
29    */
30   public ThreadedBehaviourFactory getTbf(){
31     return myTbf;
32   }
33
34   @Override
35   public int onEnd(){
36     ((SequentialBehaviour)root()).addSubBehaviour(new
          WorkerTraceCalculate(myWorkerAgent, myTarget));
37     return 0;
38   }
39
40 }//class
```

C.3.3 WorkerTraceCalculate

```
1  package kripos.geo;
2
3  import java.net.InetAddress;
4  import java.net.UnknownHostException;
5  import java.sql.Connection;
6  import java.sql.DriverManager;
7  import java.sql.ResultSet;
8  import java.sql.SQLException;
9  import java.sql.Statement;
10 import java.util.ArrayList;
11 import java.util.Date;
12 import Jama.Matrix;
13 import jade.core.behaviours.OneShotBehaviour;
14 import jade.core.behaviours.SequentialBehaviour;
15 import kripos.ontology.DNSname;
16 import kripos.ontology.IP;
17
18 public class WorkerTraceCalculate extends OneShotBehaviour {
19   private WorkerAgent myWorkerAgent;
20   private Landmark toTrace;
21   private double[] myBestLine = new double[2];
22   private ArrayList<Landmark> myLandmarks;
23   private double baseM = 0.02; //magic number due to lightspeed
         in optical fibres
24   private long allowedSlack =0;
25
26   /**
27    *
28    * @param a
29    * @param toTrace
30    */
31   public WorkerTraceCalculate(WorkerAgent a, Landmark toTrace)
       {
32     super(a);
33     myWorkerAgent = a;
34     this.toTrace = toTrace;
35   }
36
37   @Override
38   public void action() {
39     myLandmarks = myWorkerAgent.getPingList().getLandmarkList()
         ;
40
41     for(int o=0; o<myLandmarks.size();o++){
42       Landmark l = myLandmarks.get(o);
43     }
44
45     if(myWorkerAgent.getType().equalsIgnoreCase("TRACE-CBG")){
46       myBestLine = calculateBestline(myLandmarks);
47     }
48
49     Connection con = connectDB();
```

```
50      updateDB(con, myBestLine, myLandmarks);
51
52      ArrayList<String> ips =  newlyTraced(con);
53
54      if (ips.get(ips.size()-1).equalsIgnoreCase("true")){
55        toTrace = new Landmark(ips.get(0));
56        Statement st;
57        try {
58          st = con.createStatement();
59          String query2 = "SELECT␣NAME,␣MIN_RTT,␣C1,␣EPSILON,␣
              IPADR,␣CHECKED␣FROM␣TRACED␣WHERE␣IPADR␣=␣'"+ips.get
              (0)+"'";
60          ResultSet rs2 = st.executeQuery(query2);
61          while(rs2.next()){
62            double minRtt = rs2.getDouble("MIN_RTT");
63            double c1 = rs2.getDouble("C1");
64            double epsilon = rs2.getDouble("EPSILON");
65            toTrace.setMinRTT(minRtt);
66            toTrace.setC1(c1);
67            toTrace.setEpsilon(epsilon);
68          }
69          ((SequentialBehaviour)root()).addSubBehaviour(
70              new WorkerTraceFinal(myWorkerAgent, toTrace,
                  myBestLine, myLandmarks));
71
72        } catch (SQLException e) {
73          e.printStackTrace();
74        }
75      }
76      else{
77        ((SequentialBehaviour)root()).addSubBehaviour(
78            new WorkerTraceTarget(myWorkerAgent, toTrace,
                myBestLine, myLandmarks));
79      }
80    }
81
82    /**
83     * Calculates the bestline from ping measurements
84     *
85     * @param landmarks the landmarks used as a basis for
           calculating the bestline
86     * @return an array containing the two double values that
           constitutes the bestline
87     */
88    private double[] calculateBestline(ArrayList<Landmark>
         landmarks){
89      double[] result = new double[2];
90      double bestMi = 9999999;//initial nonsense values
91      double bestBi = 9999999;
92
93      double smallestY = 99999999;
94      double smallestX = 0;
95      int startAt = 1;  //requires sorted input!
96      for(int i=0;i<landmarks.size();i++){
```

```
97        double tempY = landmarks.get(i).getMinRTT();
98        //set new value for smallestY, but only if it is sane
99        if(tempY < smallestY && landmarks.get(i).getMinRTT() >=
              landmarks.get(i).getDistance() * baseM){
100         smallestY = tempY;
101         startAt = i+1;//only use landmarks with greater
                distance for calculating bestline
102         smallestX = landmarks.get(i).getDistance();
103       }
104     }
105     //set up matrices for solving y = mx + b
106     Matrix a = new Matrix(2,2);
107     Matrix b = new Matrix(2,1);
108     //fill matrices with values from landmark containing
            smallestY
109     a.set(0, 0, smallestX);
110     a.set(0, 1, 1);
111     b.set(0, 0, smallestY);
112
113     //solve y = mx + b for all the pairs (smallestY, other
            landmark)
114     for(int i=startAt;i<landmarks.size();i++){
115       a.set(1, 0, landmarks.get(i).getDistance());
116       a.set(1, 1, 1);
117
118       b.set(1, 0, landmarks.get(i).getMinRTT());
119
120       Matrix x = a.solve(b);
121       if((x.get(0,0)>baseM) && (x.get(1,0)>=0)){
122         if(x.get(0,0)< bestMi){
123           bestMi = x.get(0,0);
124           bestBi = x.get(1,0);
125         }
126       }
127     }
128
129     //choose next-smallest Y if result does not fall within
            allowed region
130     if(bestMi<baseM || bestBi<0){
131       if(landmarks.size()>4){
132         landmarks.remove(startAt-1);//remove the outlier
                measurement
133         calculateBestline(landmarks);
134       }
135       result[0] = bestMi;
136       result[1] = bestBi;
137       return result;    //TODO throw exception instead
138     }
139     else{
140       result[0] = bestMi;
141       result[1] = bestBi;
142       return result;
143     }
144   }
```

```
145
146     /**
147      * Connects to the local database and returns the Connection
             for further use.
148      *
149      * @return a connection to the database
150      */
151     private Connection connectDB(){//TODO path & password
152       Connection c = null;
153       try {
154         Class.forName("org.hsqldb.jdbcDriver");
155         c = DriverManager.getConnection("jdbc:hsqldb:hsql://
                localhost/xdb", "sa", "");
156       }catch (ClassNotFoundException e) {
157         //FIXME unable to find database classes. dosomething
158         e.printStackTrace();
159       }
160       catch (SQLException e) {
161         //FIXME unable to connect to database dosomething
162         e.printStackTrace();
163       }
164       return c;
165     }
166
167     /**
168      * Checks if an IP address has currently been traced.
169      * The definition of current is given by allowedSlack.
170      *
171      * @param con The database connection to use
172      * @return true if this IP was recently traced
173      */
174     private ArrayList<String> newlyTraced(Connection con){
175       String newlyTraced = "false";
176       ArrayList<String> returnList = new ArrayList<String>();
177       try {
178         String adrToCheck = toTrace.getIP();
179         Statement stmt = con.createStatement(ResultSet.
                TYPE_SCROLL_INSENSITIVE,ResultSet.CONCUR_READ_ONLY);
180         String query = "SELECT␣NAME,␣IPADR,␣CHECKED␣FROM␣TRACED␣
                WHERE␣IPADR␣=␣'"+adrToCheck+"'";
181         ResultSet rs = stmt.executeQuery(query);
182         if(!rs.next()){
183           returnList.add(adrToCheck);
184         }
185         rs.previous();
186         while (rs.next()) {
187           Date lastChecked  = (Date)rs.getTimestamp("CHECKED");
188           long toCheck;
189           if(rs.wasNull()){
190             toCheck = 0;
191           }
192           else{
193             toCheck = lastChecked.getTime();
194           }
```

```
195           long currentTime = myWorkerAgent.getCalendar().
                  getTimeInMillis();
196           long diff = currentTime - toCheck;
197           if(diff<allowedSlack){
198             returnList.add(rs.getString("IPADR"));
199             newlyTraced = "true";
200           }
201           else{
202             returnList.add(adrToCheck);
203           }
204         }
205       }
206       catch (SQLException e) {
207         e.printStackTrace();
208         returnList.add(newlyTraced);
209         return returnList;
210       }
211       returnList.add(newlyTraced);
212       return returnList;
213     }
214
215     /**
216      * Updates the MISC table with the new bestline information
             and
217      * any matching landmarks in the LANDMARK database with new
             minRTT, C1, Epsilon and timestamp
218      *
219      * @param con The Connection to use
220      * @param mi the m part of the new bestline
221      * @param bi the b part of the new bestline
222      */
223     private void updateDB(Connection con, double [] bestline,
            ArrayList<Landmark> newLandmarks){
224       String newBestline = "UPDATE␣MISC␣SET␣LAST_BESTLINE␣=␣now␣,
              BESTLINE_M␣=␣"+
225       bestline[0]+",␣BESTLINE_B␣="+ bestline[1];
226
227       try {
228         Statement st = con.createStatement();
229         st.executeUpdate(newBestline);
230         for(int i=0; i<newLandmarks.size();i++){
231           String landmarkUpdate = "UPDATE␣LANDMARKS␣SET␣MIN_RTT=␣
                  "
232             +newLandmarks.get(i).getMinRTT()+",␣CHECKED␣=␣now,␣C1
                    ␣=␣"+newLandmarks.get(i).getC1()+
233             ",␣EPSILON␣=␣"+newLandmarks.get(i).getEpsilon();
234           st.executeUpdate(landmarkUpdate);
235         }
236       } catch (SQLException e) {
237         e.printStackTrace();
238       }
239     }
240
241   }//class
```

C.3.4 WorkerTraceTarget

```
1  package kripos.geo;
2
3  import jade.core.behaviours.SequentialBehaviour;
4
5  import java.sql.Connection;
6  import java.sql.DriverManager;
7  import java.sql.ResultSet;
8  import java.sql.SQLException;
9  import java.sql.Statement;
10 import java.util.ArrayList;
11
12 public class WorkerTraceTarget extends PingBehave {
13   private static final long serialVersionUID =
         1898594180371430543L;
14   private WorkerAgent myWorkerAgent;
15   private double[] myBestLine;
16   private ArrayList<Landmark> myLandmarks;
17
18   public WorkerTraceTarget(WorkerAgent a, Landmark target,
         double[] bestLine,
19       ArrayList<Landmark> landmarks) {
20     super(a, target);
21     myWorkerAgent = a;
22     myBestLine = bestLine;
23     myLandmarks = landmarks;
24     myTarget = target;
25   }
26
27   public WorkerTraceTarget(WorkerAgent a, Landmark target, int
         numPings,
28       int pingDistance, boolean estimateEps, boolean estimateC1
             ,
29       double[] bestLine, ArrayList<Landmark> landmarks){
30     super(a, target, numPings, pingDistance, estimateEps,
           estimateC1);
31     myWorkerAgent = a;
32     myBestLine = bestLine;
33     myLandmarks = landmarks;
34     myTarget = target;
35   }
36
37   @Override
38   public void action() {
39     super.action();
40     updateTracedDB(connectDB(), myTarget);
41     ((SequentialBehaviour)root()).addSubBehaviour(
42         new WorkerTraceFinal(myWorkerAgent, myTarget,
               myBestLine, myLandmarks));
43   }
44
45   /**
46    * Creates a new entry in the TRACED database,
```

```
47      * or updates an existing entry if a mathcing IP address is
            found
48      *
49      * @param con The database connection to use
50      * @param toTrace the landmark containg the information to
            added
51      */
52     private void updateTracedDB(Connection con, Landmark toTrace)
            {
53       try {
54         Statement stmt = con.createStatement();
55         String query = "SELECT␣COUNT(IPADR)␣FROM␣TRACED␣WHERE␣
               IPADR='"+toTrace.getIP()+"'";
56         ResultSet rs = stmt.executeQuery(query);
57
58         while(rs.next()){
59           if(rs.getInt(1)>0){
60             String updateString = "UPDATE␣TRACED␣SET␣MIN_RTT␣="+
                   toTrace.getMinRTT()+",␣CHECKED=␣now␣WHERE␣IPADR='"
                   +toTrace.getIP()+"'";
61             stmt.executeUpdate(updateString);
62           }
63           else{
64             String ip = toTrace.getIP();
65             String name = toTrace.getName();
66             double c1 = toTrace.getC1();
67             double epsilon = toTrace.getEpsilon();
68             double minRtt = toTrace.getMinRTT();
69             String insertString = "INSERT␣INTO␣TRACED␣VALUES␣('"+
                   name+"','"+ip+"',now,"+minRtt+",null,"+
70             c1+","+epsilon+",null)";
71             stmt.executeUpdate(insertString);
72           }
73         }
74       } catch (SQLException e) {
75         e.printStackTrace();
76       }
77     }
78
79     /**
80      * Connects to the local database and returns the Connection
            for further use.
81      *
82      * @return a connection to the database
83      */
84     private Connection connectDB(){//TODO path & password
85       Connection c = null;
86       try {
87         Class.forName("org.hsqldb.jdbcDriver");
88         c = DriverManager.getConnection("jdbc:hsqldb:hsql://
               localhost/xdb", "sa", "");
89       }catch (ClassNotFoundException e) {
90         //FIXME unable to find database classes. dosomething
91         e.printStackTrace();
```

```
92      }
93      catch (SQLException e) {
94        //FIXME unable to connect to database dosomething
95        e.printStackTrace();
96      }
97      return c;
98    }
99
100 }//class
```

C.3.5 WorkerTraceFinal

```
1  package kripos.geo;
2
3  import java.sql.Connection;
4  import java.sql.DriverManager;
5  import java.sql.ResultSet;
6  import java.sql.SQLException;
7  import java.sql.Statement;
8  import java.util.ArrayList;
9  import jade.content.lang.Codec.CodecException;
10 import jade.content.lang.sl.SLCodec;
11 import jade.content.onto.OntologyException;
12 import jade.content.onto.basic.Action;
13 import jade.core.behaviours.OneShotBehaviour;
14 import jade.lang.acl.ACLMessage;
15 import kripos.ontology.DNSname;
16 import kripos.ontology.GeoLocation;
17 import kripos.ontology.IP;
18 import kripos.ontology.InternetInvestigationsOntology;
19 import kripos.ontology.TraceResult;
20 import kripos.ontology.TraceResultList;
21
22 public class WorkerTraceFinal extends OneShotBehaviour {
23   private static final long serialVersionUID =
         -6054469352616818627L;
24   private WorkerAgent myWorkerAgent;
25   private Landmark toTrace;
26   private double[] myBestLine;
27   private ArrayList<Landmark> myLandmarks;
28
29
30   public WorkerTraceFinal(WorkerAgent a, Landmark toTrace,
         double[] bestLine,
31        ArrayList<Landmark> landmarks){
32     super(a);
33     myWorkerAgent = a;
34     this.toTrace = toTrace;
35     myBestLine = bestLine;
36     myLandmarks = landmarks;
37   }
38
39   @Override
40   public void action() {
41     ACLMessage returnResult = new ACLMessage(ACLMessage.INFORM)
           ;
42     returnResult.addReceiver(myWorkerAgent.getSessionAgent());
43     returnResult.setLanguage(new SLCodec(0).getName());
44     returnResult.setOntology(InternetInvestigationsOntology.
           getInstance().getName());
45     String convID = myWorkerAgent.getConvList().
           registerConversation();
46     returnResult.setConversationId(convID);
47     Action act;
```

```
48      IP target = new IP();
49      target.setHasName(toTrace.getName());
50      GeoLocation gl = new GeoLocation();
51
52      try{
53        Connection con = connectDB();
54        Statement st = con.createStatement();
55        String getGeoLocation = "SELECT␣NAME,␣LATITUDE,␣LONGITUDE
              ␣FROM␣MISC";
56        ResultSet rs = st.executeQuery(getGeoLocation);
57        while(rs.next()){
58          Double lat = rs.getDouble("LATITUDE");
59          Double lon = rs.getDouble("LONGITUDE");
60          String name = rs.getString("NAME");
61          gl.setLocationName(name);
62          gl.setLocationLatitude(lat.floatValue());
63          gl.setLocationLongitude(lon.floatValue());
64        }
65      }
66      catch(SQLException se){
67        se.printStackTrace();
68      }
69
70      if(myWorkerAgent.getType().equalsIgnoreCase("TRACE-CBG")){
71        TraceResultList trl = new TraceResultList();
72        trl.setHasGeoLocation(gl);
73        TraceResult tr = new TraceResult();
74        tr.setAddressToBeTraced(target);//TODO only first IP
              address if DNSName
75        tr.setHasCaseID(myWorkerAgent.getCaseID());
76        tr.setHasSessionID(myWorkerAgent.getSessionID());
77        tr.setHasGeoLocation(gl);
78
79        Double geoDistance = (toTrace.getMinRTT() - myBestLine
              [1])/myBestLine[0];
80        tr.setTraceResultData(geoDistance.floatValue());
81
82        trl.addTraceResult(tr);
83        act = new Action(myWorkerAgent.getSessionAgent(), trl);
84      }
85
86      else if(myWorkerAgent.getType().equalsIgnoreCase("TRACE-
            GeoPing")){
87        TraceResultList trl = new TraceResultList();
88        trl.setHasGeoLocation(gl);
89
90        TraceResult targetTr = new TraceResult();
91        targetTr.setTraceResultData((new Double(toTrace.getMinRTT
              ()).floatValue()));
92        targetTr.setAddressToBeTraced(target);
93        trl.addTraceResult(targetTr);
94
95        for (int i=0;i<myLandmarks.size();i++){
96          TraceResult tr = new TraceResult();
```

```
97          DNSname name = new DNSname();
98          name.setHasName(myLandmarks.get(i).getName());
99          tr.setAddressToBeTraced(name);
100         tr.setTraceResultData((new Double(myLandmarks.get(i).
                getMinRTT())).floatValue());
101         trl.addTraceResult(tr);
102       }
103
104       act = new Action(myWorkerAgent.getSessionAgent(), trl);
105
106     }
107     else{
108       TraceResultList trl = new TraceResultList();
109       trl.setDoneStatus("failed");
110       act = new Action(myWorkerAgent.getSessionAgent(), trl);
111     }
112
113     try {
114       myAgent.getContentManager().fillContent(returnResult, act
              );
115       myWorkerAgent.send(returnResult);
116       myAgent.doDelete();
117
118     } catch (CodecException e) {
119       System.out.println("Sending␣of␣ACL␣message␣failed");
120       e.printStackTrace();
121     } catch (OntologyException e) {
122       System.out.println("Sending␣of␣ACL␣message␣failed");
123       e.printStackTrace();
124     }
125   }
126
127   /**
128    * Connects to the local database and returns the Connection
           for further use.
129    *
130    * @return a connection to the database
131    */
132   private Connection connectDB(){//TODO path & password
133     Connection c = null;
134     try {
135       Class.forName("org.hsqldb.jdbcDriver");
136       c = DriverManager.getConnection("jdbc:hsqldb:hsql://
              localhost/xdb", "sa", "");
137     }catch (ClassNotFoundException e) {
138       //FIXME unable to find database classes. dosomething
139       e.printStackTrace();
140     }
141     catch (SQLException e) {
142       //FIXME unable to connect to database dosomething
143       e.printStackTrace();
144     }
145     return c;
146   }
```

```
147
148 }//class
```

C.3.6 PingBehave

```
1  package kripos.geo;
2
3  import jade.core.behaviours.OneShotBehaviour;
4  import kripos.geo.ping.HostUnreachableException;
5  import kripos.geo.ping.MinRTT;
6
7  /**
8   * Performs actual pinging of remote hosts.
9   * Used by WorkerAgents and SessionAgents.
10  *
11  * @author oysteine
12  * @version 1.0
13  *
14  */
15 public class PingBehave extends OneShotBehaviour {
16   protected static final long serialVersionUID =
         9200100880741959482L;
17   protected Landmark myTarget;
18   protected AgentTemplate myAgentTemplate;
19   protected MinRTT myMinRTT;
20   protected int myNumPings = 5;
21   protected int myPingDistance = 3;
22   protected boolean estimateEps = false;
23   protected boolean estimateC1 = false;
24
25   /**
26    * Uses default values for probeCount and interprobe delay
27    * No estimation of epsilon or C1, no rePing()
28    *
29    * @param a the AgentTemplate this instance belongs to
30    * @param landmark the target to ping
31    */
32   public PingBehave(AgentTemplate a, Landmark target) {
33     super(a);
34     myAgentTemplate = a;
35     myTarget = target;
36   }
37
38   /**
39    * Uses supplied values
40    * TODO currently only IPv4
41    *
42    * @param a the AgentTemplate this instance belongs to
43    * @param landmark the target to ping
44    * @param numPings the number of pings to send
45    * @param pingDistance the time between pings are sent
46    * @param estimateEps true if epsilon is to be estimated
47    * @param estimateC1 true if C1 is to be estimated
48    */
49   public PingBehave(AgentTemplate a, Landmark target, int
         numPings,
```

```
50        int pingDistance, boolean estimateEps, boolean estimateC1
              ){
51      super(a);
52      myAgentTemplate = a;
53      myTarget = target;
54      myNumPings = numPings;
55      myPingDistance = pingDistance;
56      this.estimateEps = estimateEps;
57      this.estimateC1 = estimateC1;
58    }
59
60    /* (non-Javadoc)
61     * @see jade.core.behaviours.Behaviour#action()
62     */
63    @Override
64    public void action() {
65      myMinRTT = new MinRTT();
66      try {
67        myMinRTT.ping(myTarget.getIP(), myNumPings,
              myPingDistance, false);
68        if(estimateC1){
69          myMinRTT.computeCRegions(true);
70          myTarget.setC1(myMinRTT.getCI());
71        }
72        if(estimateEps){
73          myTarget.setEpsilon(myMinRTT.estimateEpsilon());
74        }
75        myTarget.setMinRTT(myMinRTT.getMinRTT());
76
77      } catch (HostUnreachableException e) {
78      }
79    }
80
81  }//class
```

C.4 GWAgent Classes

C.4.1 GWAgent

```
1  package kripos.gateway;
2
3  import java.util.ArrayList;
4  import java.util.GregorianCalendar;
5  import java.util.Random;
6  import java.util.concurrent.ConcurrentHashMap;
7  import jade.content.ContentElement;
8  import jade.content.lang.Codec.CodecException;
9  import jade.content.lang.sl.SLCodec;
10 import jade.content.onto.OntologyException;
11 import jade.content.onto.UngroundedException;
12 import jade.content.onto.basic.Action;
13 import jade.content.onto.basic.Result;
14 import jade.core.AID;
15 import jade.core.ContainerID;
16 import jade.domain.DFService;
17 import jade.domain.FIPAException;
18 import jade.domain.FIPAAgentManagement.DFAgentDescription;
19 import jade.domain.FIPAAgentManagement.FIPAManagementOntology;
20 import jade.domain.FIPAAgentManagement.SearchConstraints;
21 import jade.domain.FIPAAgentManagement.ServiceDescription;
22 import jade.domain.JADEAgentManagement.WhereIsAgentAction;
23 import jade.domain.mobility.MobilityOntology;
24 import jade.lang.acl.ACLMessage;
25 import jade.lang.acl.ConversationList;
26 import jade.lang.acl.MessageTemplate;
27 import jade.wrapper.gateway.GatewayAgent;
28 import kripos.ontology.InternetInvestigationsOntology;
29
30 /**
31  * @author oysteine
32  * @version 1.0
33  *
34  */
35 public class GWAgent extends GatewayAgent {
36   private static final long serialVersionUID =
         -5814890860284035720L;
37   protected ConversationList convList;
38   protected GregorianCalendar myCal;
39   protected ConcurrentHashMap<AID, String> adminAgents = new
         ConcurrentHashMap<AID, String>();
40   private ConcurrentHashMap<String, CommandPackage>
         activeCommands =
41     new ConcurrentHashMap<String, CommandPackage>();
42
43   /**
44    *
45    */
46   public GWAgent() {
```

```
47      convList = new ConversationList(this);
48      myCal = new GregorianCalendar(); //timezone, not
            implemented
49    }
50
51    @Override
52    protected void setup(){
53      getContentManager().registerLanguage(new SLCodec(0));
54      getContentManager().registerLanguage(new SLCodec());//some
            messages from AMS not set to SL-0
55      getContentManager().registerOntology(MobilityOntology.
            getInstance());
56      getContentManager().registerOntology(
            InternetInvestigationsOntology.getInstance());
57      getContentManager().registerOntology(FIPAManagementOntology
            .getInstance());
58
59      //find AdminAgents
60      DFAgentDescription adminTemplate = new DFAgentDescription()
            ;
61      ServiceDescription adminSd = new ServiceDescription();
62      adminSd.setType("AdminAgent");
63      adminTemplate.addServices(adminSd);
64      SearchConstraints sc = new SearchConstraints();
65      sc.setMaxDepth(new Long(0));
66      sc.setMaxResults(new Long(10000));
67      ArrayList<DFAgentDescription> tempList = new ArrayList<
            DFAgentDescription>();
68
69      try {
70        DFAgentDescription dfas[] = DFService.search(this,
              adminTemplate, sc);
71        for(int i=0; i<dfas.length;i++){
72          tempList.add(dfas[i]);
73        }
74
75      } catch (FIPAException e1) {
76        e1.printStackTrace();
77      }
78
79      for(int i=0;i< tempList.size();i++){
80        WhereIsAgentAction wiaa = new WhereIsAgentAction();
81        wiaa.setAgentIdentifier(tempList.get(i).getName());
82        AID myAMS = getAMS();
83        Action act = new Action(myAMS, wiaa);
84
85        ACLMessage qMsg = new ACLMessage(ACLMessage.REQUEST);
86        String convID = getName() + new Random().nextLong();
87        qMsg.setConversationId(convID);
88
89        qMsg.setLanguage(new SLCodec(0).getName());
90        qMsg.setOntology(MobilityOntology.getInstance().getName()
              );
91
```

```
92        try {
93          getContentManager().fillContent(qMsg, act);
94          qMsg.addReceiver(act.getActor());
95          //register conversation with agent to get correct
                reception
96          getConvList().registerConversation(convID);
97          send(qMsg);
98
99          MessageTemplate mt = MessageTemplate.
                MatchConversationId(convID);
100         ACLMessage resp = blockingReceive(mt, 1000000);
101         getConvList().deregisterConversation(convID);
102         ContentElement ce = getContentManager().extractContent(
                resp);
103         Result result = (Result) ce;
104         ContainerID cid = (ContainerID)result.getValue();
105         adminAgents.put(tempList.get(i).getName(), cid.getName
                ());
106       }
107       catch (UngroundedException e) {
108         e.printStackTrace();
109       } catch (CodecException e) {
110         e.printStackTrace();
111       } catch (OntologyException e) {
112         e.printStackTrace();
113       }
114     }
115
116     AID df = getDefaultDF();
117     ACLMessage adminSubs = DFService.createSubscriptionMessage(
            this, df, adminTemplate, sc);
118     convList.registerConversation(adminSubs.getConversationId()
            );
119
120     addBehaviour(new AdminSubscribeBehaviour(this, adminSubs,
            adminAgents));
121
122     addBehaviour(new GWReceiveBehaviour(this));
123   }
124
125   /**
126    *
127    */
128   @Override
129   protected void processCommand(java.lang.Object command){
130     if (command instanceof CommandPackage){
131       CommandPackage cp = (CommandPackage)command;
132       addBehaviour(new LaunchTraceBehave(this, cp));
133     }
134     else {
135       //todo throw exception to alert gw and webapp/user of
              error
136       System.out.println("erereorerer");
137       releaseCommand(command);
```

```
138       }
139     }
140
141     /**
142      * Get the CommandPackage assosciated with the conversation
             ID supplied
143      * Remove the CommandPackage from the map
144      * Deregisters the conversation ID with the agent
145      *
146      * @param conversationID to match
147      * @return commandpackage assosciated with conversationID
148      */
149     public CommandPackage getMatchingCommand(String
            conversationID){
150       convList.deregisterConversation(conversationID);
151       return activeCommands.remove(conversationID);
152     }
153
154     /**
155      * Adds an entry in the map of activeCommands
156      *
157      * @param convID
158      * @param cp
159      */
160     public void addCommand(String convID, CommandPackage cp){
161       activeCommands.put(convID, cp);
162     }
163
164     /**
165      * Returns the conversationlist of this agent
166      *
167      * @return the ConversationList of this agent
168      */
169     protected ConversationList getConvList(){
170       return convList;
171     }
172
173     /**
174      * Returns the GregorianCalendar of this agent
175      *
176      * @return the calendar of this agent
177      */
178     protected GregorianCalendar getCalendar(){
179       return myCal;
180     }
181
182   }//class
```

C.4.2 GWReceiveBehaviour

```
1  package kripos.gateway;
2
3  import java.util.ArrayList;
4
5  import jade.content.Concept;
6  import jade.content.ContentElement;
7  import jade.content.lang.Codec.CodecException;
8  import jade.content.onto.OntologyException;
9  import jade.content.onto.UngroundedException;
10 import jade.content.onto.basic.Action;
11 import jade.core.behaviours.CyclicBehaviour;
12 import jade.lang.acl.ACLMessage;
13 import jade.lang.acl.MessageTemplate;
14 import kripos.ontology.TraceResult;
15 import kripos.ontology.TraceResultList;
16 import kripos.ontology.UpdateTime;
17
18 /**
19  * @author oysteine
20  * @version 1.0
21  *
22  */
23 public class GWReceiveBehaviour extends CyclicBehaviour {
24   private static final long serialVersionUID =
         2014289984738025171L;
25   private GWAgent myGWAgent;
26
27   /**
28    * @param a
29    */
30   public GWReceiveBehaviour(GWAgent a) {
31     super(a);
32     myGWAgent = a;
33   }
34
35   /* (non-Javadoc)
36    * @see jade.core.behaviours.Behaviour#action()
37    */
38   @Override
39   public void action() {
40     MessageTemplate mt = myGWAgent.getConvList().
           getMessageTemplate();
41     ACLMessage msg = myAgent.receive(mt);
42     if(msg != null){
43       handle(msg);
44     }
45     block(100); //this SHOULD work without timeout
46     //block();
47   }
48
49   /**
50    * Handles incoming messages.
```

```
51      * If the received action is not understood a NOT_UNDERSTOOD
            messages is returned to the sender.
52      *
53      * @param msg incoming {@link jade.lang.acl.ACLMessage}
54      */
55     private void handle(ACLMessage msg){
56       try {
57         ContentElement content = myAgent.getContentManager().
               extractContent(msg);
58         Concept action = ((Action)content).getAction();
59
60         if (action instanceof TraceResultList){
61           TraceResultList trl = (TraceResultList)action;
62           CommandPackage cp = myGWAgent.getMatchingCommand(msg.
                 getConversationId());
63           ArrayList<TraceResult> resList = new ArrayList<
                 TraceResult>();
64           for(int i=0;i<trl.getTraceResult().size();i++){
65             resList.add((TraceResult)trl.getTraceResult().get(i))
                   ;
66           }
67           if(cp.getType().equalsIgnoreCase("TRACE-CBG")){
68             cp.setCBGResults(resList);
69
70           }
71           else if (cp.getType().equalsIgnoreCase("TRACE-GEOPING")
                 ){
72             cp.setGeoPingResults(resList);
73           }
74
75           cp.setRepTime(myGWAgent.getCalendar().getTime());
76           if(trl.getDoneStatus().equalsIgnoreCase("OK")){
77             cp.setSuccessful(true);}
78           else{
79             cp.setSuccessful(false);
80           }
81           myGWAgent.releaseCommand(cp);
82         }
83         else if (action instanceof UpdateTime){
84           UpdateTime  upd = (UpdateTime)((Action)content).
                 getAction();
85           long offset = Math.round(upd.getHasOffset());
86           myGWAgent.getCalendar().setTimeInMillis(offset +
                 myGWAgent.getCalendar().getTimeInMillis());
87           //log to SysLog
88         }
89         else {
90           ACLMessage reply = msg.createReply();
91           msg.setPerformative(ACLMessage.NOT_UNDERSTOOD);
92           myAgent.send(reply);
93         }
94       }
95       catch (UngroundedException e) {
96         e.printStackTrace();
```

```
97        }
98
99        catch (CodecException e) {
100         e.printStackTrace();
101       }
102
103       catch (OntologyException e) {
104         e.printStackTrace();
105       }
106     }
107
108 }//class
```

C.4.3 LaunchTraceBehave

```
1  package kripos.gateway;
2
3  import java.math.BigInteger;
4  import java.util.GregorianCalendar;
5  import java.util.Iterator;
6  import java.util.Set;
7  import jade.content.lang.Codec.CodecException;
8  import jade.content.lang.sl.SLCodec;
9  import jade.content.onto.OntologyException;
10 import jade.content.onto.basic.Action;
11 import jade.core.AID;
12 import jade.core.behaviours.SimpleBehaviour;
13 import jade.domain.mobility.MobilityOntology;
14 import jade.lang.acl.ACLMessage;
15 import jade.lang.acl.MessageTemplate;
16 import kripos.ontology.*;
17
18 /**
19  * Launched by GWAgent when a CommandPackage containing
        instructions to commence a trace is received
20  *
21  * @author oysteine
22  * @version 1.0
23  */
24 public class LaunchTraceBehave extends SimpleBehaviour {
25   private static final long serialVersionUID =
        4144187902498096653L;
26   private CommandPackage command;
27   private GWAgent myGWAgent;
28   private boolean done;
29
30   /**
31    * Creates an instance of this behaviour
32    *
33    * @param a the GWAgent owning this instance
34    * @param the CommandPackage containing information about
         what to do
35    */
36   public LaunchTraceBehave(GWAgent a, CommandPackage cp) {
37     super(a);
38     command = cp;
39     myGWAgent = a;
40   }
41
42   /* (non-Javadoc)
43    * @see jade.core.behaviours.Behaviour#action()
44    */
45   @Override
46   public void action() {
47     //parse package
48     command.getReqTime();
49     command.getTarget();
```

```
50      command.getType();
51
52      //construct and fill CreateCase, Case and Session
53      CreateCase cc = new CreateCase();
54      OntAddress target;
55      if(isIP()){
56        IP ip = new IP();
57        ip.setHasName(command.getTarget());
58        target = ip;
59      }
60      else{
61        DNSname dns = new DNSname();
62        dns.setHasName(command.getTarget());
63        target = dns;
64      }
65      Case c = new Case();
66      c.setCaseID((int)Math.round((Math.random())*100000000));
67      c.setCaseName("testcase"); //todo get from GUI
68
69      Session s = new Session();
70      s.setCaseID(c.getCaseID());
71
72      s.setHasBaseAddress(target);
73      c.setCaseSessions(s);
74      cc.setHasCase(c);
75
76      OntDate date = new OntDate();
77      long time = GregorianCalendar.getInstance().getTimeInMillis
            ();
78      date.setTime(time);
79      cc.setHasDate(date);
80      date.setTime(time+10000);
81      c.setCaseStartDate(date);
82
83      Action act = new Action(getRandomAdminAgent(),cc);
84      String convID = sendRequest(act);
85      myGWAgent.addCommand(convID, command);
86      MessageTemplate mt = MessageTemplate.MatchConversationId(
            convID);
87      myAgent.blockingReceive(mt);
88    }
89
90    /**
91     *
92     * @return true if the String represents and IP address and
           not DNS name
93     */
94    private boolean isIP(){
95      String toParse = command.getTarget().replace('.', '0');
96      try{
97        new BigInteger(toParse);
98        return true;
99      }
100     catch(NumberFormatException nfe){
```

```
101         return false;
102       }
103     }
104
105     /**
106      * Get a random AdminAgent from the hashmap of AdminAgents
107      *
108      * @return the AID of the selected AdminAgent
109      */
110     private AID getRandomAdminAgent(){
111       int size = myGWAgent.adminAgents.size();
112       int random = (int)Math.round((Math.random())*size);
113       Set<AID> tempSet = myGWAgent.adminAgents.keySet();
114       Iterator<AID> it = tempSet.iterator();
115       for(int i=0;i<random-1;i++){
116         it.next();
117       }
118       return it.next();
119     }
120
121     /**
122      * Utility method for sending ACLMessages
123      * <p>
124      * Registers the conversationID in the convList
125      * to avoid loosing any reply
126      *
127      * @param action the action that is to be wrapped in an
           ACLMessage
128      * @return the conversationID for the created ACLMessage
129      */
130     private String sendRequest(Action action){
131       try {
132         ACLMessage qMsg = new ACLMessage(ACLMessage.REQUEST);
133         String convID = myGWAgent.getConvList().
              registerConversation();
134         qMsg.setConversationId(convID);
135         qMsg.setLanguage(new SLCodec(0).getName());
136         qMsg.setOntology(MobilityOntology.getInstance().getName()
              );
137         myAgent.getContentManager().fillContent(qMsg, action);
138         qMsg.addReceiver(action.getActor());
139         //register conversation with agent to get correct
              reception of reply
140         myAgent.send(qMsg);
141         return convID;
142       }
143       catch (CodecException e) {
144         e.printStackTrace();
145         return null;
146       }
147       catch (OntologyException e) {
148         e.printStackTrace();
149         return null;
150       }
```

```
151     }
152
153     /* (non-Javadoc)
154      * @see jade.core.behaviours.Behaviour#done()
155      */
156     @Override
157     public boolean done() {
158       return done;
159     }
160
161   }//class
```

C.4.4 AdminSubscribeBehaviour

```
1  package kripos.gateway;
2
3  import java.util.concurrent.ConcurrentHashMap;
4  import jade.content.ContentElement;
5  import jade.content.lang.Codec.CodecException;
6  import jade.content.lang.sl.SLCodec;
7  import jade.content.onto.OntologyException;
8  import jade.content.onto.UngroundedException;
9  import jade.content.onto.basic.Action;
10 import jade.content.onto.basic.Result;
11 import jade.core.AID;
12 import jade.core.ContainerID;
13 import jade.domain.DFService;
14 import jade.domain.FIPAException;
15 import jade.domain.FIPAAgentManagement.DFAgentDescription;
16 import jade.domain.JADEAgentManagement.WhereIsAgentAction;
17 import jade.domain.mobility.MobilityOntology;
18 import jade.lang.acl.ACLMessage;
19 import jade.lang.acl.MessageTemplate;
20 import jade.proto.SubscriptionInitiator;
21
22 /**
23  * Subscribes to changes in registered AdminAgents by the AMS
       Service.
24  * Any changes are recorded in a ConcurrentHashMap
25  * that was first initialized by setup().
26  *
27  * @author oysteine
28  * @version 1.0
29  */
30 public class AdminSubscribeBehaviour extends
       SubscriptionInitiator {
31
32   private static final long serialVersionUID =
         9887486124117325 43L;
33   private GWAgent myAgent;
34   private ConcurrentHashMap<AID, String> myMap;
35
36   /**
37    * Constructor
38    *
39    * @param a The agent this instance belongs to
40    * @param msg The message used by the behaviour to initiate
         subscription
41    * @param map The hashmap for recording changes.
42    */
43   public AdminSubscribeBehaviour(GWAgent a, ACLMessage msg,
         ConcurrentHashMap<AID, String> map) {
44     super(a, msg);
45     myAgent = a;
46     myMap = map;
47   }
```

```
48
49    /**
50     * Handles incoming inform messages that contains changes in
          registered AdminAgents.
51     * The AMS is queried for the Location of any new AdminAgents
          .
52     * Removed AdminAgents are delete from the hashmap.
53     */
54    @Override
55    protected void handleInform(ACLMessage inform) {
56      try {
57        DFAgentDescription[] dfds =
58          DFService.decodeNotification(inform.getContent());
59        for(int i=0; i<dfds.length;i++){
60          AID aid = dfds[i].getName();
61
62          if(myMap.containsKey(aid)){
63            /* There seems to be a problem with the AMS, messages
                  containing
64             * removal information are sent without reason. This
                  functionality disabled.
65              myMap.remove(aid);
66             */
67          }
68          else{
69            WhereIsAgentAction wiaa = new WhereIsAgentAction();
70            wiaa.setAgentIdentifier(aid);
71            AID myAMS = myAgent.getAMS();
72            Action act = new Action(myAMS, wiaa);
73
74            ACLMessage qMsg = new ACLMessage(ACLMessage.REQUEST);
75            String convID = myAgent.getConvList().
                registerConversation();
76            qMsg.setConversationId(convID);
77
78            qMsg.setLanguage(new SLCodec(0).getName());
79            qMsg.setOntology(MobilityOntology.getInstance().
                getName());
80            try {
81              myAgent.getContentManager().fillContent(qMsg, act);
82              qMsg.addReceiver(act.getActor());
83              //register conversation with agent to get correct
                    reception
84              myAgent.send(qMsg);
85
86              MessageTemplate mt = MessageTemplate.
                    MatchConversationId(convID);
87              ACLMessage resp = myAgent.blockingReceive(mt,
                    1000000);
88              myAgent.getConvList().deregisterConversation(convID
                    );
89              ContentElement ce = myAgent.getContentManager().
                    extractContent(resp);
90              Result result = (Result) ce;
```

```
91              ContainerID cid = (ContainerID)result.getValue();
92              myMap.put(aid, cid.getName());
93
94            } catch (UngroundedException e) {
95              e.printStackTrace();
96            } catch (CodecException e) {
97              e.printStackTrace();
98            } catch (OntologyException e) {
99              e.printStackTrace();
100           }
101         }
102       }
103     }
104     catch (FIPAException fe) {
105       fe.printStackTrace();
106     }
107   }
108
109 }//class
```

C.4.5 CommandPackage

```
1  //part of design
2
3  package kripos.gateway;
4
5  import java.util.Date;
6  import java.util.ArrayList;
7
8  import kripos.geo.DelayVector;
9  import kripos.geo.Landmark;
10 import kripos.ontology.TraceResult;
11
12 /**
13  * Class for wrapping all necessary information in a bundle for
        easy exchange between
14  * the multi-agent system and the gateway-classes.
15  *
16  * @author oysteine
17  * @version 1.0
18  */
19 public class CommandPackage {
20   private String target; //the target of the trace
21   private String type; //trace type. may be extended to also
        cover securing of content etc
22   private Date reqTime;// when was the trace requested
23   private Date repTime;// when was the trace finalized
24   private boolean successful = false; //the final state of the
        trace
25   private ArrayList<Landmark> geoPingResults;
26   private ArrayList<TraceResult> CBGResults  = new ArrayList<
        TraceResult>();
27   private ArrayList<DelayVector> delayVectors = new ArrayList<
        DelayVector>();
28
29   /**
30    * Construct an empty CommandPackage.
31    * Any fields must be filled by the relevant set methods.
32    */
33   public CommandPackage() {
34   }
35
36   public void addResult(TraceResult tr){
37     CBGResults.add(tr);
38   }
39
40   /**
41    *
42    * @return the cBGResults
43    */
44   public ArrayList<TraceResult> getCBGResults() {
45     return CBGResults;
46   }
47
```

```
48
49      /**
50       * @param results the cBGResults to set
51       */
52      public void setCBGResults(ArrayList<TraceResult> results) {
53        CBGResults = results;
54      }
55
56
57      /**
58       * @return the geoPingResults
59       */
60      public ArrayList getGeoPingResults() {
61        return geoPingResults;
62      }
63
64
65      /**
66       * @param geoPingResults the geoPingResults to set
67       */
68      public void setGeoPingResults(ArrayList geoPingResults) {
69        this.geoPingResults = geoPingResults;
70      }
71
72
73      /**
74       * @return the repTime
75       */
76      public Date getRepTime() {
77        return repTime;
78      }
79
80
81      /**
82       * @param repTime the repTime to set
83       */
84      public void setRepTime(Date repTime) {
85        this.repTime = repTime;
86      }
87
88
89      /**
90       * @return the reqTime
91       */
92      public Date getReqTime() {
93        return reqTime;
94      }
95
96
97      /**
98       * @param reqTime the reqTime to set
99       */
100     public void setReqTime(Date reqTime) {
101       this.reqTime = reqTime;
```

```
102     }
103
104
105     /**
106      * @return the successful
107      */
108     public boolean isSuccessful() {
109       return successful;
110     }
111
112
113     /**
114      * @param successful the successful to set
115      */
116     public void setSuccessful(boolean successful) {
117       this.successful = successful;
118     }
119
120
121     /**
122      * @return the target
123      */
124     public String getTarget() {
125       return target;
126     }
127
128
129     /**
130      * @param target the target to set
131      */
132     public void setTarget(String target) {
133       this.target = target;
134     }
135
136
137     /**
138      * @return the type
139      */
140     public String getType() {
141       return type;
142     }
143
144
145     /**
146      * @param type the type to set
147      */
148     public void setType(String type) {
149       this.type = type;
150     }
151
152     public ArrayList<DelayVector> getDelayVectors(){
153       return delayVectors;
154     }
155
```

```
156 }//class
```

C.5 Ping Classes

C.5.1 MinRTT

```
1  package kripos.geo.ping;
2  //TODO support ping4 and ping6 simultaneously
3  /*
4   * Some of the functionality of this class is adapted from
       RTTOmeter.
5   * (Code originally released under GPLv2)
6   * The RTTOmeter application is available from: http://idmaps.
       eecs.umich.edu/rttometer/
7   * The GPLv2 is available at: http://www.gnu.org/copyleft/gpl.
       html
8   **/
9
10 import java.io.BufferedReader;
11 import java.io.IOException;
12 import java.io.InputStream;
13 import java.io.InputStreamReader;
14 import java.nio.charset.Charset;
15 import java.util.ArrayList;
16 import java.util.Arrays;
17
18 /**
19  *
20  * @author oysteine
21  * @version 1.0
22  *
23  */
24 public class MinRTT {
25   private float CI = 0;
26   private float CII = 0;
27   private       float CIII = 0;
28   private ArrayList<PingItem> probeList = new ArrayList<
       PingItem>();
29   private float minimumRTT = 99999.0f;
30   private float maximumRTT = 0;
31   private float epsilon = MINEPS;//automatically changed based
       on minimumRTT
32
33   /* default values of epsilon */
34   public final static float UNINETTEPS = 0.7f; //single AS,
       most paths <<20ms
35   public final static float MINEPS = 2;  //<=50ms
36   public final static float MEDEPS = 4;  //50-150ms
37   public final static float BIGEPS = 6;  //>150ms
38
39   private float CI_threshold = 0.8f; //confidence in CI
40   private final int EPS_MIN_PROBES = 100; //magic number. only
       for measuring between landmarks
41   private boolean estimate_CI_confidence;
42
```

```
43    private int listCounter = 0; //used to resume calculations if
          the probeList is extended
44    private float accCi = 0; //cumulative
45    private float accCii = 0; //cumulative
46    private float accCiii = 0; //cumulative
47    private int accN = 0; //cumulative
48
49    private int numProbes; //Updated if successive runtimes
50    private int rePingCnt = 0;
51    //used by rePing()
52    private String previousAddress;
53    private int previousNumPings;
54    private double previousPingDistance;
55
56    /**
57     *
58     */
59    public MinRTT(){
60    }
61
62    /**
63     *
64     * @param address
65     * @param numPings
66     * @param pingDistance
67     * @throws HostUnreachableException
68     */
69    public ArrayList ping(String address, int numPings, double
          pingDistance,
70        boolean reset) throws HostUnreachableException{
71      if(reset){
72        reset();
73      }
74
75      previousAddress = address;
76      previousNumPings = numPings;
77      previousPingDistance = pingDistance;
78
79      int lastSequence = 0;
80      listCounter = 0;
81
82      try {
83        Runtime runtime = Runtime.getRuntime();
84        String command = "ping␣-i"+pingDistance+"␣-c"+numPings+"␣
              " + address;
85        Process p = runtime.exec(command);
86        numProbes += numPings;
87        InputStream ip = p.getInputStream();
88        InputStreamReader isr = new InputStreamReader(ip, Charset
              .forName("ISO-8859-1"));
89        BufferedReader br = new BufferedReader(isr);
90
91        String temp = null;
92        while((temp = br.readLine()) !=null){
```

```
93          //handle unreachable host
94          if(temp.contains("0␣received,␣100%␣packet␣loss")){
95            for(int i=0;i<numPings;i++){
96              probeList.add( new PingItem(PingItem.LOST,
                    lastSequence+i+1,-999999));
97              listCounter++;
98            }
99            throw new HostUnreachableException("100%␣packet␣loss"
                  );
100         }
101
102         //sort out filtered replies
103         if(temp.contains("Packet␣filtered")){
104           int cutSub = temp.indexOf("icmp_seq=");
105           String subTemp = temp.substring(cutSub);
106           int cutSeq = subTemp.indexOf('=');
107           int cutSeqEnd = subTemp.indexOf("Packet␣filtered");
108           String seqString = subTemp.substring(cutSeq+1,
                  cutSeqEnd-1);
109           int seq = Integer.parseInt(seqString);
110
111           int seqDelta = seq - lastSequence;
112           if(seqDelta > 1){
113             for(int i=0; i<seqDelta;i++){
114               PingItem lostPi = new PingItem(PingItem.LOST,
                      lastSequence+i,-999999);
115               probeList.add(lostPi);
116               listCounter++;
117             }
118           }
119
120           lastSequence = seq;
121           PingItem pi = new PingItem(PingItem.FILTERED, seq,
                  -999999);
122           probeList.add(pi);
123           listCounter++;
124           throw new HostUnreachableException("Packet␣filtering␣
                  on␣path␣to␣host");
125         }
126
127         //normal pingitem, if not 64 bytes it is not a regular
128         //ICMP_ECHO_REPLY packet and we ignore it
129         if(temp.contains("64␣bytes␣from")){
130           int cutPoint = temp.indexOf("=");
131           String cutTemp = temp.substring(cutPoint);
132           int seqEndPoint = cutTemp.indexOf("ttl");
133           String seqString = cutTemp.substring(1,seqEndPoint-1)
                  ;
134           int seq = Integer.parseInt(seqString);
135
136           //adds missing probes as they are not listed by ping'
                  s output
137           int seqDelta = seq - lastSequence;
138           if(seqDelta > 1){
```

```
139             for(int i=0; i<seqDelta;i++){
140               PingItem lostPi = new PingItem(PingItem.LOST,
                      lastSequence+i,-999999);
141               probeList.add(lostPi);
142               listCounter++;
143             }
144           }
145
146           lastSequence = seq;
147           int timePoint = temp.lastIndexOf("=");
148           int endPoint = temp.lastIndexOf("ms");
149           String time = temp.substring(timePoint+1, endPoint-1)
                  ;
150           float rtt = Float.parseFloat(time);
151
152           PingItem pi = new PingItem(PingItem.NORMAL_RTT, seq,
                  rtt);
153           probeList.add(pi);
154           listCounter++;
155         }
156
157         if(temp.contains("rtt␣min/avg/max/mdev␣=␣")){
158
159           //common variables
160           int cutPoint = temp.indexOf("=");
161           String cutTemp = temp.substring(cutPoint+2);
162           int firstSlashPoint = cutTemp.indexOf("/");
163
164           //maxRTT  //not needed
165           int maxTempEndPointLong = cutTemp.indexOf("ms");
166           String maxTempLong = cutTemp.substring(0,
                  maxTempEndPointLong);
167           int maxTempEndPoint = maxTempLong.lastIndexOf("/");
168           String maxTemp = maxTempLong.substring(0,
                  maxTempEndPoint);
169           int maxEndPoint = maxTemp.lastIndexOf("/");
170           int maxPoint = maxTemp.indexOf("/");
171           String max = maxTemp.substring(maxPoint+1,
                  maxEndPoint);
172           float maxRtt = Float.parseFloat(max);
173           if (maximumRTT < maxRtt){
174             maximumRTT = maxRtt;
175           }
176
177           //minRTT
178           String min = cutTemp.substring(0, firstSlashPoint);
179           float minRtt = Float.parseFloat(min);
180           if(minimumRTT > minRtt){
181             minimumRTT = minRtt;
182           }
183         }//if
184
185       }//while
186     }//try
```

```
187       catch (IOException e) {
188         throw new HostUnreachableException(e.getMessage(),e.
              getCause());
189       }
190       return probeList;
191     }
192
193     /**
194      *
195      *
196      * Method adapted from RTTOmeter application
197      */
198     public int computeCRegions(boolean estimateConfidence){
199       estimate_CI_confidence = estimateConfidence;
200       float ci = 0;
201       float cii = 0;
202       float ciii = 0;
203       int n = 0; //total number of phaseplot points (not probes)
204       float val = 0;
205       float w1 = 0;
206       float w2 = 0;
207       float w = 0;
208       int loopStart = 0;
209
210       if(minimumRTT <20){
211         epsilon = UNINETTEPS;
212       }
213       else if(minimumRTT <50){
214         epsilon = MINEPS;
215       }
216       else if (minimumRTT <150){
217         epsilon = MEDEPS;
218       }
219       else{
220         epsilon = BIGEPS;
221       }
222
223       if(probeList.size() != listCounter){
224         loopStart = (probeList.size() - listCounter)-1; //include
                the last probe from the previous run
225       }
226
227       for(int i=loopStart;i<probeList.size()-1;i++) {//we do not
            want to compare the last pingitem to null
228         PingItem piCurrent = probeList.get(i);
229         PingItem piNext = probeList.get(i+1);
230
231         //Classify the point in the phase-plot
232         if(piCurrent.getType() == PingItem.NORMAL_RTT && piNext.
              getType() == PingItem.NORMAL_RTT ){
233           n++;
234
235           /* Point in CI, CII, or CIII */
236           if( (piCurrent.getRTT() - minimumRTT) <= epsilon){
```

```
237             w1 = 1.0f;
238           }
239           else {
240             val = ((int)((piCurrent.getRTT() - minimumRTT)/
                    epsilon) + 1);
241             w1 = 1.0f/val;
242           }
243
244           if( (piNext.getRTT() - minimumRTT) <= epsilon){
245             w2 = 1.0f;
246           }
247           else {
248             val = ((int)((piNext.getRTT() - minimumRTT)/epsilon)
                    + 1);
249             w2 = 1.0f/val;
250           }
251
252           w = w1 * w2;
253         }
254         else {
255           /* Point in CII or CIII */
256           if(piCurrent.getType() == PingItem.LOST) {
257             w1 = 0.0f;
258           } else {
259             val = ((int)((piCurrent.getRTT() - minimumRTT)/
                    epsilon) + 1);
260             w1 = 1.0f/val;
261           }
262
263           if(piNext.getType() == PingItem.LOST) {
264             w2 = 0.0f;
265           } else {
266             val = ((int)((piNext.getRTT() - minimumRTT)/epsilon)
                    + 1);
267             w2 = 1.0f/val;
268           }
269
270           w = w1 * w2;
271         }
272
273         ci += w;
274         if(w1 == 1 || w2 == 1){
275           /* This point in CII */
276           cii += (1-w);
277         }
278         else{
279           /* Definitely CIII */
280           ciii += (1-w);
281         }
282       }
283       accCi=+ci;
284       accCii=+cii;
285       accCiii=+ciii;
286       accN=+n;
```

```
287
288       if(accN > 0) {
289         CI = accCi/accN;
290         CII =  accCii/accN;
291         CIII =  accCiii/accN;
292       }
293       if(estimate_CI_confidence && CI< CI_threshold && rePingCnt
              <= 2){
294         rePing();
295       }
296       return n;
297
298     }//compute
299
300     /**
301      *
302      *
303      */
304     private void rePing() {//reuses original input. TODO:estimate
          better values. wait() if c2 high?
305       rePingCnt++;
306       try {
307         ping(previousAddress, previousNumPings,
              previousPingDistance, false);
308         computeCRegions(estimate_CI_confidence);
309       } catch (HostUnreachableException e) {
310         // TODO Auto-generated catch block
311         e.printStackTrace();
312       }
313     }
314
315     private void reset(){
316       accCi = 0;
317       accCii = 0;
318       accCiii = 0;
319       accN = 0;
320       rePingCnt = 0;
321       numProbes = 0;
322       CI = 0;
323       CII = 0;
324       CIII = 0;
325       probeList.clear();
326       minimumRTT = 99999.0f;
327       maximumRTT = 0;
328     }
329
330     /**
331      * Estimates epsilon from the ping measurements
332      *
333      * Method adapted from RTTOmeter application
334      */
335     public float estimateEpsilon(){
336       int n = 0;
337       float eps = -1.0f;
```

```
338       float[] rtts = new float[numProbes];
339
340       /* make array of rtts from probelist */
341       for(int i=0;i<probeList.size();i++) {
342         PingItem piCurrent = probeList.get(i);
343         if(piCurrent.getType() == PingItem.NORMAL_RTT ){
344           rtts[n++] = piCurrent.getRTT();
345         }
346       }
347
348       if (n >= (0.8 * EPS_MIN_PROBES)) { /* At least 80% of the
              probes are successful */
349         Arrays.sort(rtts);
350         ModeNode mn = new ModeNode();
351         float mode_all = mn.mode(rtts, n);
352
353         /* Estimate epsilon */
354         eps = 2*(mode_all - rtts[0]); //or use minimumRTT, same
              value
355
356         /* Make sure eps >=0, mode may be very close to min,
357          * and due to precision the subtraction might give
               negative result
358          */
359         if(eps < 0){
360           eps = 0.0f;
361         }
362       }
363       return eps;
364     }
365
366     /*
367      * Getter methods
368      */
369
370     public float getCI(){
371       return CI;
372     }
373
374     public float getCII(){
375       return CII;
376     }
377
378     public float getCIII(){
379       return CIII;
380     }
381
382     public float getMaxRTT(){
383       return maximumRTT;
384     }
385
386     public float getMinRTT(){
387       return minimumRTT;
388     }
```

```
389
390 }//class
```

C.5.2 ModeNode

```
1  package kripos.geo.ping;
2  /*
3   * Most of the functionality of this class is adapted from
        RTTOmeter.
4   * While the main RTTOmeter application is released under the
        GPLv2,
5   * the file mode.c is apparently not (breach of GPL?).
6   * The notice below is required by the original author.
7   */
8
9  /*
10  * Copyright (c) 2003, Amgad Zeitoun.
11  * All rights reserved.
12  *
13  * Redistribution and use in source and binary forms are
        permitted
14  * provided that the above copyright notice and this paragraph
        are
15  * duplicated in all such forms and that any documentation,
16  * advertising materials, and other materials related to such
17  * distribution and use acknowledge that the software was
        developed
18  * by Amgad Zeitoun at the University of Michigan, Ann Arbor.
        The
19  * name of the University may not be used to endorse or promote
20  * products derived from this software without specific prior
21  * written permission.
22  * THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS
        OR
23  * IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE
        IMPLIED
24  * WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR
        PURPOSE.
25  *
26  * Author:
27  *           Amgad Zeitoun (azeitoun@eecs.umich.edu)
28  */
29
30 import java.util.Hashtable;
31
32 public class ModeNode {
33   private Long key;
34   private long cnt;
35
36   /**
37    *
38    * @param x
39    * @return
40    */
41   private int ceiling(float x){
42     if( (int)(Math.abs(x) + 0.5) > (int)(Math.abs(x)) ){
43       return (int)(Math.abs(x+0.5));
```

```
44      }
45      else {
46        return (int)(Math.abs(x));
47      }
48    }
49
50    /**
51     *
52     * @param m1
53     * @param m2
54     * @return
55     */
56    public int matchNode(ModeNode m1, ModeNode m2){
57      if(m1.getKey() < m2.getKey()) {
58        return -1;
59      }
60
61      if(m1.getKey() > m2.getKey()) {
62        return 1;
63      }
64      return 0;
65    }
66
67
68    /**
69     *    Calaculate the mode of an array of values with length
          len
70     * @param array
71     * @param len
72     * @return
73     */
74    public float mode(float[] array, int len){
75      float mode = -1.0f;
76      long max_cnt = 0;
77      long val = 0;
78      boolean dbl_precision;
79      long key;
80
81      if (array.length <= 0){
82        return mode;
83      }
84      Hashtable<Long,ModeNode> hashtable = new Hashtable<Long,
          ModeNode>(array.length);
85
86      /* If the RTTs are very small (i.e., less than 1 ms), make
          double digitis mode
87                    precision, otherwise make it a single digit
                          precision */
88      /* NOTE: I assume that array is sorted. Which is true,
          because I call mode()
89                    after I qsort the array
90       */
91
92      if (array[0] < 1.0){
```

```
93        dbl_precision = true;
94      }
95      else{
96        dbl_precision = false;
97      }
98
99      for (float element : array) {
100
101       /* Using a single digit precision by default, except when
              the values
102                       of RTTs are really small (<1.0ms) */
103       if(dbl_precision){
104         key = ceiling(element * 100);
105       }
106       else{
107         key = ceiling(element * 10);
108       }
109
110       ModeNode m = hashtable.get(key);
111       if(m == null) {
112         m = new ModeNode();
113         m.setKey(key);
114         hashtable.put(m.getKey(), m);
115       }
116
117       m.incCnt();
118
119       if(m.getCnt() > max_cnt) {
120         /* we have a mode here */
121         max_cnt = m.getCnt();
122         val = key;
123       }
124       /* TODO: detect multiple modes */ //dette fikser vi
125     }
126
127     /* The mode should be repeated more than once! */
128     /* Just in case we don't have any mode at all */
129     if ( max_cnt > 1 ) {
130       if(dbl_precision){
131         mode = (float)val/100;
132       }
133       else{
134         mode = (float)val/10;
135       }
136     }
137     return mode;
138
139   }//mode
140
141   /*
142    * Setter and getter methods
143    */
144
145   public void incCnt(){
```

```
146       cnt++;
147     }
148
149
150     public long getCnt(){
151       return cnt;
152     }
153
154
155     public Long getKey(){//param?
156       return key;
157     }
158
159
160     public void setKey(Long newKey){
161       key = newKey;
162     }
163
164   }//class
```

C.5.3 PingItem

```
1  package kripos.geo.ping;
2
3  public class PingItem {
4    public final static int FILTERED=-2;
5    public final static int LOST=-1;
6    public final static int NORMAL_RTT=0;
7    public final static int MIN_RTT=1;//not needed
8    public final static int MAX_RTT=2;//not needed
9    public final static int AVG_RTT=3;//not needed
10   public final static int MED_RTT=4;//not needed
11   private final int type;
12   private final int sequence;
13   private final float rtt;
14
15   /**
16    *
17    * @param newType
18    * @param seqnr
19    * @param rtt
20    */
21   public PingItem(int newType, int seqnr, float rtt){
22     type = newType;
23     sequence = seqnr;
24     this.rtt = rtt;
25   }
26
27   /*
28    * Getter methods
29    */
30
31   /**
32    * The type indicates if this PingItem represents
33    * real ping information or a lost packet
34    *
35    * @return the type of this PingItem
36    */
37   public int getType(){
38     return type;
39   }
40
41   /**
42    * The sequence number only has local meaning.
43    *
44    * @return the sequence number of this PingItem
45    */
46   public int getSequence(){
47     return sequence;
48   }
49
50   /**
51    *
52    * @return the RTT of this PingItem
```

```
53      */
54     public float getRTT(){
55       return rtt;
56     }
57 
58 }//class
```

C.5.4 HostUnreachableException

```
1  package kripos.geo.ping;
2
3  public class HostUnreachableException extends Exception {
4
5    private static final long serialVersionUID =
         -4310128500792634318L;
6
7    public HostUnreachableException() {
8    }
9
10   /**
11    *
12    * @param message
13    */
14   public HostUnreachableException(String message) {
15     super(message);
16   }
17
18   /**
19    *
20    * @param cause
21    */
22   public HostUnreachableException(Throwable cause) {
23     super(cause);
24   }
25
26   /**
27    *
28    * @param message
29    * @param cause
30    */
31   public HostUnreachableException(String message, Throwable
         cause) {
32     super(message, cause);
33   }
34
35 }//class
```

C.6 Servlet Classes

C.6.1 GetTrace

```
1  package kripos.math.servlet;
2
3
4
5  import java.io.*;
6  import javax.servlet.*;
7  import javax.servlet.http.*;
8
9  /**
10  * Display a page where an IP to trace can be enter.
11  * Display a map with trace information if an IP has been
        entered.
12  *
13  * @author oysteine
14  */
15 public class GetTrace extends HttpServlet {
16
17   private static final long serialVersionUID =
         6720480250134837969L;
18
19
20   @Override
21   public void doGet(HttpServletRequest request,
         HttpServletResponse response)
22   throws IOException, ServletException
23   {
24     String ip = request.getParameter("ip");
25
26     response.setContentType("text/html");
27     PrintWriter out = response.getWriter();
28     out.println("<html>");
29     out.println("<head>");
30     out.println("<title>Trace␣IP</title>");
31     out.println("</head>");
32     out.println("<body>");
33
34     out.println("<h3>Perform␣tracing</h3>");
35     out.println("<form␣action=\"Map\"␣method=POST>");
36     out.println("IP:");
37     out.print("<input␣type=text␣size=20␣name=ip");
38     if (ip != null) {
39       out.print("␣value=\"" + ip + "\"");
40     }
41     out.println(">");
42     out.println("<br>");
43     out.println("<input␣type=submit␣name=action␣value=\"Get␣
           trace\">␣");
44     out.println("<input␣type=submit␣name=action␣value=\"Force␣
           trace\">");
```

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

C.6.2 GetMap

```
1  package kripos.math.servlet;
2
3
4  import java.io.ByteArrayOutputStream;
5  import java.io.File;
6  import java.io.IOException;
7  import java.io.PrintStream;
8  import java.io.PrintWriter;
9  import java.util.Locale;
10
11 import javax.servlet.ServletContext;
12 import javax.servlet.ServletException;
13 import javax.servlet.http.HttpServlet;
14 import javax.servlet.http.HttpServletRequest;
15 import javax.servlet.http.HttpServletResponse;
16
17 import kripos.geo.openmap.IntersectionInfo;
18 import kripos.geo.openmap.MapDrawer;
19
20 /**
21  * Display a page with the results of a trace.
22  * The trace is performed if not cached already.
23  *
24  * @author oysteine
25  */
26 public class GetMap extends HttpServlet {
27   private static final long serialVersionUID =
         7631170050282471534L;
28
29   private static String BASE_PATH = null;
30
31
32   /**
33    * @return path for OpenMap data. Does <i>not</i> end with
          separator.
34    */
35   public static String getDataPath() {
36     String path = BASE_PATH;
37     if (path == null) {
38       path = System.getProperty("user.dir") + File.separator +
             "data";
39     }
40     return path;
41   }
42
43
44
45   @Override
46   public void doGet(HttpServletRequest request,
         HttpServletResponse response)
47   throws IOException, ServletException
48   {
```

```
49      ServletContext sc = getServletContext();
50      if (BASE_PATH == null) {
51        BASE_PATH = sc.getRealPath("") + File.separator + "data";
52      }
53
54      String ip = request.getParameter("ip");
55      String cacheId = request.getParameter("cacheId");
56
57      String strZoom = request.getParameter("zoom");
58      String strLongOffset = request.getParameter("longOffset");
59      String strLatOffset = request.getParameter("latOffset");
60      float zoom = 1.0f;
61      if (strZoom != null) {
62        zoom = Float.parseFloat(strZoom);
63      }
64      double longOffset = 0.0;
65      if (strLongOffset != null) {
66        longOffset = Double.parseDouble(strLongOffset);
67      }
68      double latOffset = 0.0;
69      if (strLatOffset != null) {
70        latOffset = Double.parseDouble(strLatOffset);
71      }
72
73
74
75      String action = request.getParameter("action");
76      boolean forceTrace = false;
77      String retrieval;
78      if ("Force␣trace".equals(action)) {
79        forceTrace = true;
80        retrieval = "Forced␣new␣trace";
81      } else if ("Get␣trace".equals(action)) {
82        if (TraceCache.hasTraceId(ip)) {
83          retrieval = "Using␣cached␣trace";
84        } else {
85          retrieval = "Performed␣new␣trace";
86        }
87      } else if ("Zoom␣in".equals(action)) {
88        retrieval = "Refreshed␣map␣of␣previous␣trace";
89      } else if ("Zoom␣out".equals(action)) {
90        retrieval = "Refreshed␣map␣of␣previous␣trace";
91      } else if ("Move␣up".equals(action)) {
92        retrieval = "Refreshed␣map␣of␣previous␣trace";
93      } else if ("Move␣down".equals(action)) {
94        retrieval = "Refreshed␣map␣of␣previous␣trace";
95      } else if ("Move␣left".equals(action)) {
96        retrieval = "Refreshed␣map␣of␣previous␣trace";
97      } else if ("Move␣right".equals(action)) {
98        retrieval = "Refreshed␣map␣of␣previous␣trace";
99      } else {
100       sc.log("Unknown␣action␣'" + action + "'");
101       response.setStatus(HttpServletResponse.
              SC_INTERNAL_SERVER_ERROR);
```

```
102         return;
103       }
104
105       if (ip == null) {
106         sc.log("Destination␣not␣specified!");
107         response.setStatus(HttpServletResponse.
              SC_INTERNAL_SERVER_ERROR);
108         return;
109       }
110
111       // Perform trace or retrieve cached trace
112       if (cacheId == null) {
113         cacheId = TraceCache.getTraceId(ip, forceTrace);
114       }
115       TraceCacheEntry ce = TraceCache.getCachedTrace(cacheId, ip)
            ;
116       if (ce == null) {
117         sc.log("Could␣not␣find␣cache␣id␣'" + cacheId + "'␣for␣
              address␣'" + ip + "'.");
118         response.setStatus(HttpServletResponse.
              SC_INTERNAL_SERVER_ERROR);
119         return;
120       }
121
122       IntersectionInfo intersectionInfo = MapDrawer.
            getIntersectionInfo(ce.cacheId, ce.circles);
123       //try { Thread.sleep(500); } catch (Exception e) {}
124
125
126       response.setContentType("text/html");
127       PrintWriter out = response.getWriter();
128       out.println("<html>");
129       out.println("<head>");
130       out.println("<title>Trace␣of␣" + ip + "</title>");
131       out.println("</head>");
132       out.println("<body>");
133
134       out.println("<h3>Trace␣info</h3>");
135       out.println("Retrieval:␣" + retrieval + "<br>");
136       out.println("IP␣traced:␣" + ip + "<br>");
137       out.println("Trace␣started:␣" + ce.start + "<br>");
138       out.println("Trace␣time:␣" + ce.runTime + "<br>");
139       out.printf("<i>The␣trace␣information␣is␣cached␣for␣%d␣hours
            ␣" +
140           "after␣last␣usage</i><br>%n",
141           (TraceCache.CACHE_LIFE_TIME / 3600000));
142       out.println("<h3>Trace␣results</h3>");
143       out.printf(Locale.US,
144           "Estimated␣confidence␣region:␣%,.0f␣km<sup>2</sup><br>%
                n",
145           intersectionInfo.polygonArea);
146       out.println("Estimated␣location:␣");
147       if (!intersectionInfo.centroidAvailable) {
148         out.println("(not␣available)");
```

```
149     } else {
150       out.println("longitude=" + degToString(intersectionInfo.
              centroidLongitude) +
151           ",␣latitude=" + degToString(intersectionInfo.
                  centroidLatitude) +
152       "<br>");
153     }
154     out.println("<table><tr>");
155     out.println("<td><form␣action=\"Map\"␣method=POST>");
156     out.print("<input␣type=hidden␣name=ip␣value=\"" + ip + "\">
            ");
157     out.print("<input␣type=hidden␣name=cacheId␣value=\"" +
            cacheId + "\">");
158     out.print("<input␣type=hidden␣name=zoom␣value=\"" + (zoom *
             2) + "\">");
159     out.print("<input␣type=hidden␣name=longOffset␣value=\"" + (
            longOffset) + "\">");
160     out.print("<input␣type=hidden␣name=latOffset␣value=\"" + (
            latOffset) + "\">");
161     out.println("<input␣type=submit␣name=action␣value=\"Zoom␣in
            \">");
162     out.println("</form></td>");
163     out.println("<td><form␣action=\"Map\"␣method=POST>");
164     out.print("<input␣type=hidden␣name=ip␣value=\"" + ip + "\">
            ");
165     out.print("<input␣type=hidden␣name=cacheId␣value=\"" +
            cacheId + "\">");
166     out.print("<input␣type=hidden␣name=zoom␣value=\"" + (zoom /
             2) + "\">");
167     out.print("<input␣type=hidden␣name=longOffset␣value=\"" + (
            longOffset) + "\">");
168     out.print("<input␣type=hidden␣name=latOffset␣value=\"" + (
            latOffset) + "\">");
169     out.println("<input␣type=submit␣name=action␣value=\"Zoom␣
            out\">");
170     out.println("</form></td>");
171     out.println("<td><form␣action=\"Map\"␣method=POST>");
172     out.print("<input␣type=hidden␣name=ip␣value=\"" + ip + "\">
            ");
173     out.print("<input␣type=hidden␣name=cacheId␣value=\"" +
            cacheId + "\">");
174     out.print("<input␣type=hidden␣name=zoom␣value=\"" + (zoom)
            + "\">");
175     out.print("<input␣type=hidden␣name=longOffset␣value=\"" + (
            longOffset) + "\">");
176     out.print("<input␣type=hidden␣name=latOffset␣value=\"" + (
            latOffset+2) + "\">");
177     out.println("<input␣type=submit␣name=action␣value=\"Move␣up
            \">");
178     out.println("</form></td>");
179     out.println("<td><form␣action=\"Map\"␣method=POST>");
180     out.print("<input␣type=hidden␣name=ip␣value=\"" + ip + "\">
            ");
```

```
181     out.print("<input␣type=hidden␣name=cacheId␣value=\"" +
            cacheId + "\">");
182     out.print("<input␣type=hidden␣name=zoom␣value=\"" + (zoom)
            + "\">");
183     out.print("<input␣type=hidden␣name=longOffset␣value=\"" + (
            longOffset) + "\">");
184     out.print("<input␣type=hidden␣name=latOffset␣value=\"" + (
            latOffset-2) + "\">");
185     out.println("<input␣type=submit␣name=action␣value=\"Move␣
            down\">");
186     out.println("</form></td>");
187     out.println("<td><form␣action=\"Map\"␣method=POST>");
188     out.print("<input␣type=hidden␣name=ip␣value=\"" + ip + "\">
            ");
189     out.print("<input␣type=hidden␣name=cacheId␣value=\"" +
            cacheId + "\">");
190     out.print("<input␣type=hidden␣name=zoom␣value=\"" + (zoom)
            + "\">");
191     out.print("<input␣type=hidden␣name=longOffset␣value=\"" + (
            longOffset-2) + "\">");
192     out.print("<input␣type=hidden␣name=latOffset␣value=\"" + (
            latOffset) + "\">");
193     out.println("<input␣type=submit␣name=action␣value=\"Move␣
            left\">");
194     out.println("</form></td>");
195     out.println("<td><form␣action=\"Map\"␣method=POST>");
196     out.print("<input␣type=hidden␣name=ip␣value=\"" + ip + "\">
            ");
197     out.print("<input␣type=hidden␣name=cacheId␣value=\"" +
            cacheId + "\">");
198     out.print("<input␣type=hidden␣name=zoom␣value=\"" + (zoom)
            + "\">");
199     out.print("<input␣type=hidden␣name=longOffset␣value=\"" + (
            longOffset+2) + "\">");
200     out.print("<input␣type=hidden␣name=latOffset␣value=\"" + (
            latOffset) + "\">");
201     out.println("<input␣type=submit␣name=action␣value=\"Move␣
            right\">");
202     out.println("</form></td>");
203     out.println("</tr></table>");
204     out.println("<img␣src=\"MapImage?" +
205         "ip=" + ip +
206         "&cacheId=" + cacheId +
207         "&zoom=" + zoom +
208         "&longOffset=" + longOffset +
209         "&latOffset=" + latOffset +
210         "\"" +
211         "␣border=1" +
212         "␣width=" + intersectionInfo.imageWidth +
213         "␣height=" + intersectionInfo.imageHeight +
214     "␣alt=\"OpenMap(tm)␣image\"><br>");
215
216     out.println("</body>");
217     out.println("</html>");
```

```
218     }
219
220     @Override
221     public void doPost(HttpServletRequest request,
            HttpServletResponse response)
222     throws IOException, ServletException
223     {
224       doGet(request, response);
225     }
226
227
228     private String degToString(double degrees) {
229       int deg = (int)degrees;
230       double minutes = (degrees - deg) * 60;
231
232       ByteArrayOutputStream tmp = new ByteArrayOutputStream();
233       PrintStream stream = new PrintStream(tmp);
234       stream.printf(Locale.US, "%d&deg;␣%.2f'", deg, minutes);
235       return tmp.toString();
236     }
237
238   }
```

C.6.3 GetMapImage

```
1  package kripos.math.servlet;
2
3
4  import java.io.IOException;
5  import java.io.OutputStream;
6
7  import javax.servlet.ServletContext;
8  import javax.servlet.ServletException;
9  import javax.servlet.http.HttpServlet;
10 import javax.servlet.http.HttpServletRequest;
11 import javax.servlet.http.HttpServletResponse;
12
13 import kripos.geo.openmap.MapDrawer;
14
15 /**
16  * Create a map image from the cached circles corresponding to
17  * the given ID, using OpenMap.
18  *
19  * @author oysteine
20  */
21 public class GetMapImage extends HttpServlet {
22
23
24   private static final long serialVersionUID =
         6720480250134837969L;
25
26
27   @Override
28   public void doGet(HttpServletRequest request,
         HttpServletResponse response)
29   throws IOException, ServletException
30   {
31     ServletContext sc = getServletContext();
32
33     String ip = request.getParameter("ip");
34     String cacheId = request.getParameter("cacheId");
35     TraceCacheEntry ce = TraceCache.getCachedTrace(cacheId, ip)
           ;
36     if (ce == null) {
37       sc.log("Could␣not␣find␣cache␣id␣'" + cacheId + "'␣for␣
             address␣'" + ip + "'.");
38       response.setStatus(HttpServletResponse.
             SC_INTERNAL_SERVER_ERROR);
39       return;
40     }
41
42
43     String strZoom = request.getParameter("zoom");
44     String strLongOffset = request.getParameter("longOffset");
45     String strLatOffset = request.getParameter("latOffset");
46     float zoom = 1.0f;
47     if (strZoom != null) {
```

```
48        zoom = Float.parseFloat(strZoom);
49      }
50      double longOffset = 0.0;
51      if (strLongOffset != null) {
52        longOffset = Double.parseDouble(strLongOffset);
53      }
54      double latOffset = 0.0;
55      if (strLatOffset != null) {
56        latOffset = Double.parseDouble(strLatOffset);
57      }
58
59
60      byte[] image = getImage(ce, zoom, longOffset, latOffset);
61      String filename = "mapView.gif";
62
63      String mimeType = sc.getMimeType(filename);
64      if (mimeType == null) {
65        sc.log("Could not get MIME type of " + filename);
66        response.setStatus(HttpServletResponse.
              SC_INTERNAL_SERVER_ERROR);
67        return;
68      }
69
70      response.setContentType(mimeType);
71      response.setContentLength(image.length);
72
73      OutputStream out = response.getOutputStream();
74      out.write(image);
75      out.close();
76    }
77
78    @Override
79    public void doPost(HttpServletRequest request,
          HttpServletResponse response)
80    throws IOException, ServletException
81    {
82      doGet(request, response);
83    }
84
85
86    private byte[] getImage(TraceCacheEntry ce,
87        float zoom, double longOffset, double latOffset) {
88      return MapDrawer.getMap(ce.cacheId, ce.circles,
89          zoom, latOffset, longOffset);
90    }
91
92  }
```

C.6.4 TraceCache

```
1  package kripos.math.servlet;
2
3  import jade.util.BasicProperties;
4  import jade.wrapper.ControllerException;
5  import jade.wrapper.StaleProxyException;
6  import jade.wrapper.gateway.JadeGateway;
7
8  import java.util.ArrayList;
9  import java.util.Date;
10 import java.util.HashMap;
11 import java.util.Iterator;
12 import java.util.LinkedList;
13 import java.util.List;
14 import java.util.Map;
15 import java.util.Random;
16
17 import kripos.gateway.CommandPackage;
18 import kripos.math.circle.Circle;
19 import kripos.ontology.GeoLocation;
20 import kripos.ontology.TraceResult;
21
22 /**
23  * Perform IP tracing and add the results to cache.
24  *
25  * @author oysteine
26  */
27 public class TraceCache {
28
29
30   /** Max life time of a cache entry in milliseconds. */
31   static final long CACHE_LIFE_TIME = 24 * 3600 * 1000L;
32
33   static private final Random rand = new Random();
34
35
36   private final static Object cacheMutex = new Object();
37
38   /** Mapping from cacheId. */
39   private final static Map<Long,TraceCacheEntry> idMap =
40     new HashMap<Long,TraceCacheEntry>();
41
42   /**
43    * Mapping from IP to list of cacheId for all results
44    * performed for within cache life time.
45    * Most recent trace is stored first in the list. */
46   private final static Map<String,List<Long>> destinationMap =
47     new HashMap<String,List<Long>>();
48
49
50   /**
51    * Check if <code>destination</code> is cached.
52    *
```

```
53      * @param ip
54      * @return <code>true</code> iff <code>destination</code> can
55      *  be retrieved with {@link #getTraceId(String, boolean)}
56      *  without performing a trace
57      *
58      */
59     static boolean hasTraceId(String destination) {
60       synchronized (cacheMutex) {
61         List<Long> tmp = destinationMap.get(destination);
62         if (tmp != null) {
63           Long id = tmp.get(0);
64           TraceCacheEntry ce = idMap.get(id);
65           ce.lastAccess = new Date(); // to make sure it's still
                 available when retrieving it a little later
66           return true;
67         } else {
68           return false;
69         }
70       }
71     }
72     /**
73      * Perform a trace or return most recent previous trace of <
           code>ip</code>,
74      * if one has been done.
75      *
76      * @param ip a valid IP address
77      * @param forceNew if <code>true</code> a new trace will
           always be performed,
78      *          even if <code>ip</code> already exists in the
           cache
79      * @return a cache id that can be used with {@link #
           getCachedTrace(String,String)}
80      */
81     static String getTraceId(String destination, boolean forceNew
        ) {
82       cleanCache();
83
84       Long id = null;
85       if (!forceNew) {
86         synchronized (cacheMutex) {
87           List<Long> tmp = destinationMap.get(destination);
88           if (tmp != null) {
89             id = tmp.get(0);
90           }
91         }
92       }
93
94       if (id == null) {
95         // NOTE: it is possible that two traces are performed
               with the same IP
96         //       at the same time
97         TraceCacheEntry ce = performTrace(destination);
98         insertToCache(ce);
99         id = ce.cacheId;
```

```
100       }
101
102       return id.toString();
103     }
104
105
106     /**
107      * Retrieve the trace specified by <code>cacheId</code> from
           cache.
108      *
109      * @param cacheId     an id retrieved by {@link #getTrace(
           String, boolean)} recently
110      * @param destination used to assert that the <code>cacheId</
           code>
111      *          entry was performed with this destination.
112      *          If it was not, the request is treated as if <code
           >cacheId</code>
113      *          was not found
114      * @return <code>null</code> if <code>cacheId</code> is not
           malformed,
115      *          it does not exist, or
116      *          it does not match <code>destination</code>
117      *
118      */
119     static TraceCacheEntry getCachedTrace(String cacheId, String
          destination) {
120       cleanCache();
121
122       Long id = null;
123       try {
124         id = Long.parseLong(cacheId);
125       } catch (NumberFormatException e) {
126         return null;
127       }
128
129       TraceCacheEntry ce;
130       synchronized (cacheMutex) {
131         ce = idMap.get(id);
132       }
133
134       if (ce == null || !ce.destination.equals(destination)) {
135         return null;
136       }
137
138       return ce;
139     }
140
141
142     static private void insertToCache(TraceCacheEntry ce) {
143       synchronized (cacheMutex) {
144         long id = -1;
145         synchronized (rand) {
146           while (id < 0 || idMap.containsKey(id)) {
147             id = rand.nextLong();
```

```
148         }
149       }
150       ce.cacheId = id;
151
152       idMap.put(ce.cacheId, ce);
153
154       List<Long> tmp = destinationMap.get(ce.destination);
155       if (tmp == null) {
156         tmp = new LinkedList<Long>();
157         destinationMap.put(ce.destination, tmp);
158       }
159       tmp.add(0, ce.cacheId);
160     }
161   }
162
163   /**
164    * Remove entries from cache if they are older than
165    * {@linkplain #CACHE_LIFE_TIME} ms.
166    */
167   static private void cleanCache() {
168     synchronized (cacheMutex) {
169       long curTime = System.currentTimeMillis();
170       for (Map.Entry<String,List<Long>> entry : destinationMap.
              entrySet()) {
171         List<Long> idList = entry.getValue();
172         Iterator<Long> iter = idList.iterator();
173         while (iter.hasNext()) {
174           Long id = iter.next();
175           TraceCacheEntry ce = idMap.get(id);
176           if (curTime - ce.lastAccess.getTime() >
                  CACHE_LIFE_TIME) {
177             idMap.remove(id);
178             iter.remove();
179           }
180         }
181         if (idList.size() == 0) {
182           destinationMap.remove(entry.getKey());
183         }
184       }
185     }
186   }
187
188
189
190   /**
191    * @param destination a unique destination identifier
192    * @return resulting circles from a trace
193    */
194   static private TraceCacheEntry performTrace(String
          destination) {
195     long startTime = System.currentTimeMillis();
196     List<Circle> circles = TRACELIBRARY_getTrace(destination);
197     long endTime = System.currentTimeMillis();
198     return new TraceCacheEntry(
```

```
199           destination, circles,
200           new Date(startTime), endTime - startTime);
201     }
202
203
204     /**
205      * Interfaces with the multi-agent system to perfrom a trace
206      *
207      * @param the IP address to trace
208      * @return the geographical constraint circles
209      */
210     static private List<Circle> TRACELIBRARY_getTrace(String
            destination) {
211       double KMperDegree = 1/111.15;
212       List<Circle> circles = new LinkedList<Circle>();
213
214       BasicProperties prop = new BasicProperties();
215       prop.setProperty("platform-id",     "futurum01.item.ntnu.no
              :1099/JADE");//hard coded for now :(
216       prop.setBooleanProperty("main", true);
217       prop.setProperty("mainURL","http://futurum01.item.ntnu.no
              :7778/acc");//hard coded for now :(
218       JadeGateway.init("kripos.gateway.GWAgent", prop);
219       CommandPackage cp = new CommandPackage();
220       cp.setTarget(destination);
221       cp.setType("TRACE-CBG");
222
223       try {
224
225         JadeGateway.execute(cp);
226       } catch (StaleProxyException e) {
227         e.printStackTrace();
228       } catch (ControllerException e) {
229         e.printStackTrace();
230       } catch (InterruptedException e) {
231         e.printStackTrace();
232       }
233
234       ArrayList<TraceResult> traceRes = cp.getCBGResults();
235       for(TraceResult tr : traceRes){
236         GeoLocation geoLoc = tr.getHasGeoLocation();
237         double lat = geoLoc.getLocationLatitude();
238         double lon = geoLoc.getLocationLongitude();
239         double radius = tr.getTraceResultData() * KMperDegree;
240         circles.add(new Circle(lat, lon, radius));
241       }
242
243       return circles;
244     }
245
246 }//class
```

C.6.5 TraceCacheEntry

```
1  package kripos.math.servlet;
2
3
4  import java.util.Date;
5  import java.util.List;
6
7  import kripos.math.circle.Circle;
8
9  class TraceCacheEntry {
10
11   /** Address that was traced. */
12   final String destination;
13   /** Result of trace. */
14   final List<Circle> circles;
15   /** Time trace was started. */
16   final Date start;
17   /** Milliseconds used to perform the trace. */
18   final long runTime;
19
20   /** Last time this cache entry was used. */
21   Date lastAccess;
22
23   /**
24    * Internal {@link TraceCache} cache id.
25    * Set before added to the cache.
26    */
27   long cacheId = -1;
28
29   /**
30    * You must set {@link #cacheId} before inserting this entry
          to cache
31    *
32    * @param destination
33    * @param circles
34    */
35   TraceCacheEntry(
36       String destination, List<Circle> circles,
37       Date start, long runTime) {
38     this.destination = destination;
39     this.circles = circles;
40     this.start = start;
41     this.runTime = runTime;
42
43     this.lastAccess = new Date();
44   }
45
46 }
```

C.6.6 MapDrawer

```
1  package kripos.geo.openmap;
2
3
4  import java.awt.Color;
5  import java.awt.Point;
6  import java.awt.Shape;
7  import java.awt.geom.Area;
8  import java.awt.geom.FlatteningPathIterator;
9  import java.awt.geom.GeneralPath;
10 import java.awt.geom.PathIterator;
11 import java.awt.geom.Point2D;
12 import java.util.HashMap;
13 import java.util.Iterator;
14 import java.util.LinkedList;
15 import java.util.List;
16 import java.util.Map;
17 import java.util.Properties;
18 import java.util.concurrent.locks.ReentrantLock;
19
20 import kripos.math.circle.Circle;
21 import kripos.math.circle.Intersection;
22 import kripos.math.circle.Intersector;
23 import kripos.math.servlet.GetMap;
24
25 import com.bbn.openmap.LatLonPoint;
26 import com.bbn.openmap.Layer;
27 import com.bbn.openmap.MapBean;
28 import com.bbn.openmap.event.LayerStatusEvent;
29 import com.bbn.openmap.event.LayerStatusListener;
30 import com.bbn.openmap.image.AcmeGifFormatter;
31 import com.bbn.openmap.layer.OMGraphicHandlerLayer;
32 import com.bbn.openmap.layer.location.LocationHandler;
33 import com.bbn.openmap.layer.location.LocationLayer;
34 import com.bbn.openmap.layer.location.csv.CSVLocationHandler;
35 import com.bbn.openmap.layer.shape.ShapeLayer;
36 import com.bbn.openmap.omGraphics.OMCircle;
37 import com.bbn.openmap.omGraphics.OMGraphic;
38 import com.bbn.openmap.omGraphics.OMGraphicList;
39 import com.bbn.openmap.omGraphics.OMPoint;
40 import com.bbn.openmap.omGraphics.OMPoly;
41 import com.bbn.openmap.proj.Length;
42 import com.bbn.openmap.proj.Mercator;
43 import com.bbn.openmap.proj.Proj;
44 import com.bbn.openmap.proj.Projection;
45
46 /**
47  * Draw circles on a map.
48  * @author oysteine
49  */
50 public class MapDrawer {
51
52   private static final int MAX_CACHE = 100;
```

```
53    private static final Object cacheMutex = new Object();
54    private static final Map<Long,CacheEntry> mapCache =
55      new HashMap<Long,CacheEntry>();
56
57    private static final List<Long> cacheLru = new LinkedList<
          Long>();
58
59    private static class CacheEntry {
60      private final Long cacheId;
61      private final ReentrantLock lock = new ReentrantLock();
62      /** Set <i>after</i> all other data have been set. */
63      private MapBean mapBean = null;
64      private LatLonPoint pointUpperLeft;
65      private LatLonPoint pointLowerRight;
66      private LatLonPoint pointCenter;
67      private float scale;
68      private float origScale;
69
70      private double area;
71      private LatLonPoint centroid = null;
72
73      private LayerListener listener;
74      private LocationHandler locationHandler;
75
76      private CacheEntry(Long cacheId) {
77        this.cacheId = cacheId;
78      }
79    }
80
81    private static CacheEntry acquireMap(Long cacheId) {
82      CacheEntry ce;
83      synchronized (cacheMutex) {
84        ce = mapCache.get(cacheId);
85        if (ce == null) {
86          ce = unsafeNewCacheEntry(cacheId);
87        }
88
89        cacheLru.remove(cacheId);
90        cacheLru.add(cacheId);
91      }
92
93      ce.lock.lock();
94      return ce;
95    }
96
97    private static void releaseMap(CacheEntry ce) {
98      synchronized (cacheMutex) {
99        ce.lock.unlock();
100       cacheLru.add(ce.cacheId);
101     }
102   }
103
104
105   private static CacheEntry unsafeNewCacheEntry(Long cacheId) {
```

```
106       assert Thread.holdsLock(cacheMutex);
107
108       CacheEntry ce = new CacheEntry(cacheId);
109       mapCache.put(cacheId, ce);
110       if (mapCache.size() > MAX_CACHE) {
111         mapCache.remove(cacheLru.remove(0));
112       }
113       return ce;
114     }
115
116
117
118     /**
119      *
120      * @param cacheId use cached drawing instead of creating
             using <code>circles</code>,
121      *          if the cache holds anything
122      * @param circles
123      * @return informat about area and centroid that will be
             drawn on map
124      */
125     static public IntersectionInfo getIntersectionInfo(
126         Long cacheId, List<Circle> circles) {
127
128       CacheEntry ce = acquireMap(cacheId);
129       try {
130         if (ce.mapBean == null) {
131           createMap(ce, circles);
132         }
133
134         if (ce.centroid == null) {
135           return new IntersectionInfo(
136               ce.area,
137               ce.mapBean.getWidth(),
138               ce.mapBean.getHeight());
139         } else {
140           return new IntersectionInfo(
141               ce.area,
142               ce.centroid.getLatitude(),
143               ce.centroid.getLongitude(),
144               ce.mapBean.getWidth(),
145               ce.mapBean.getHeight());
146         }
147
148       } finally {
149         releaseMap(ce);
150       }
151     }
152
153
154
155     private static final Proj dummyProj =
156       new Mercator(new LatLonPoint(0.123456, -0.123456), 1.42E7f,
              640, 480);
```

```
157
158
159     /**
160      * @param cacheId use cached drawing instead of creating
             using <code>circles</code>,
161      *          if the cache holds anything
162      * @param circles
163      * @param zoom 1.0 is with polygon extremes on the borders,
164      *          smaller values are larger portion of the world,
165      *          greater values are zoomed inside polygon
166      * @param latOffset north offset of quasi center of polygon
             in decimal degrees
167      * @param longOffset east offset of quasi center of polygon
             in decimal degrees
168      * @return a map with <code>circles</code> painted on it
169      */
170     static public byte[] getMap(Long cacheId, List<Circle>
            circles,
171         float zoom, double latOffset, double longOffset) {
172
173       CacheEntry ce = acquireMap(cacheId);
174       try {
175         if (ce.mapBean == null) {
176           createMap(ce, circles);
177         }
178
179
180
181         LatLonPoint customCenter = new LatLonPoint(
182             ce.pointCenter.getLatitude() + latOffset,
183             ce.pointCenter.getLongitude() + longOffset);
184         float customScale = ce.scale / zoom;
185         if (customScale > ce.origScale) {
186           customScale = ce.origScale;
187         }
188
189         // disable city names if showing too much of world at
                once
190         ce.locationHandler.setShowNames(customScale < 2.5E7);
191
192         // update map
193         Proj proj = new Mercator(customCenter, customScale,
194             ce.mapBean.getWidth(), ce.mapBean.getHeight());
195
196         // if we use only default parameters the first time, then
                for some reason:
197         // - traceLayer is not drawn
198         // - repaints hang in LayerListener
199         // so we use a dummy projection first
200         ce.listener.resetCompletion(1);
201         ce.mapBean.setProjection(dummyProj);
202         ce.listener.waitForCompletion();
203
204         ce.listener.resetCompletion(1);
```

```
205         ce.mapBean.setProjection(proj);
206         ce.listener.waitForCompletion();
207 
208         // create image
209         AcmeGifFormatter gifFormatter = new AcmeGifFormatter();
210         byte[] image = gifFormatter.getImageFromMapBean(ce.
                mapBean);
211         return image;
212 
213       } finally {
214         releaseMap(ce);
215       }
216     }
217 
218 
219 
220 
221     private static void createMap(CacheEntry ce, List<Circle>
          circles) {
222       String shapeFile = GetMap.getDataPath() + "/shape/dcwpo-
            browse.shp";
223       String spatialFile = GetMap.getDataPath() + "/shape/dcwpo-
            browse.ssx";
224       String locationFile = GetMap.getDataPath() + "/cities.csv";
225 
226       ce.listener = new LayerListener();
227 
228       MapBean mapBean = new MapBean();
229       mapBean.setSize(1024, 768);
230       MapBean.suppressCopyright = true; // suppress after first
            use
231       // political borders
232       ShapeLayer backgroundLayer = new ShapeLayer();
233       ce.listener.addLayer(backgroundLayer, 1);
234       Properties backgroundProps = new Properties();
235       backgroundProps.put("prettyName", "Political␣Solid");
236       backgroundProps.put("lineColor", "000000");
237       backgroundProps.put("fillColor", "BDDE83");
238       backgroundProps.put("shapeFile", shapeFile);
239       backgroundProps.put("spatialIndex", spatialFile);
240       backgroundLayer.setProperties(backgroundProps);
241       assert mapBean.getComponentCount() == 0 : mapBean.
            getComponentCount();
242       mapBean.add(backgroundLayer, 0);
243 
244       OMGraphicList traceItems = new OMGraphicList(2);
245       OMGraphicList tracePolygon = new OMGraphicList(1);
246       OMGraphicList traceCircles = new OMGraphicList(circles.size
            ());
247 
248       // inaccurate intersections used for initial scale
            calculation
249       List<Intersection> inters2D = Intersector.getIntersections(
            circles);
```

```
250       inters2D = Intersector.getMorePoints(inters2D, 0.2);
251
252       setBorders(ce, inters2D, traceItems, mapBean);
253       setBorderInfo(ce, traceItems);
254
255       setPolygon2D(inters2D, tracePolygon);
256       setCircles(circles, traceCircles);
257
258       OMGraphicHandlerLayer traceLayer = new
              OMGraphicHandlerLayer();
259       ce.listener.addLayer(traceLayer, 1);
260       traceItems.add(tracePolygon);
261       traceItems.add(traceCircles);
262       traceLayer.setList(traceItems);
263       mapBean.add(traceLayer, 0);
264       ce.listener.waitForCompletion();
265
266       List<Point2D> screenPath = getWindingPath(ce, traceCircles,
              mapBean);
267       Point centroidXY = getScreenCentroid(screenPath);
268       if (centroidXY == null) {
269         ce.centroid = null;
270       } else {
271         ce.centroid = mapBean.getProjection().inverse(centroidXY)
              ;
272       }
273       List<LatLonPoint> approximatedPath = new LinkedList<
              LatLonPoint>();
274       for(Point2D next : screenPath){
275         Point temp = new Point((int)next.getX(), (int)next.getY()
              );
276         approximatedPath.add(mapBean.getProjection().inverse(temp
              ));
277       }
278       ce.area = getApproximatedPolygonArea(approximatedPath);
279
280       traceItems = new OMGraphicList(2);
281       tracePolygon = new OMGraphicList(1);
282       traceCircles = new OMGraphicList(circles.size());
283
284       setBorderInfo(ce, traceItems);
285       setApproximatedPolygon(approximatedPath, tracePolygon);
286       setCircles(circles, traceCircles);
287
288       LocationLayer locations = new LocationLayer();
289       ce.listener.addLayer(locations, 1);
290       ce.locationHandler = new CSVLocationHandler();
291       Properties lhProps = new Properties();
292       lhProps.put("locationFile", locationFile);
293       lhProps.put("csvFileHasHeader", "false");
294       lhProps.put("showNames", "true");
295       //lhProps.put("nameColor", "008C54");
296       lhProps.put("nameColor", "000000");
297       lhProps.put("showLocations", "true");
```

```
298       lhProps.put("locationColor", "FF0000");
299
300       lhProps.put("name.lineColor", "FF008C54");
301       lhProps.put("location.lineColor", "FFFF0000");
302       lhProps.put("location.fillColor", "FFaaaaaa");
303       lhProps.put("location.pointRadius", "2");
304       lhProps.put("location.pointOval", "true");
305
306       lhProps.put("nameIndex", "0");
307       lhProps.put("latIndex", "2");
308       lhProps.put("lonIndex", "3");
309       ce.locationHandler.setProperties(lhProps);
310
311       //locations.setProperties(lhProps);
312       locations.setLocationHandlers(new LocationHandler[] { ce.
              locationHandler });
313
314       Proj proj = new Mercator(ce.pointCenter, ce.scale,
315           mapBean.getWidth(), mapBean.getHeight());
316       ce.listener.resetCompletion(1);
317       traceItems.add(tracePolygon);
318       traceItems.add(traceCircles);
319       traceLayer.setList(traceItems);
320       mapBean.add(locations, 0);
321       mapBean.setProjection(proj);
322       ce.listener.waitForCompletion();
323
324       ce.mapBean = mapBean;
325     }
326
327     private static double getApproximatedPolygonArea(List<
              LatLonPoint> sortedPath){
328       if (sortedPath.size() < 3) {
329         return 0.0;
330       }
331
332       List<LatLonPoint> tmp = new LinkedList<LatLonPoint>(
              sortedPath);
333       LatLonPoint base = tmp.remove(0);
334
335       int triangleCount = 0;
336       double interiorAngleSum = 0.0;
337       LatLonPoint prev = tmp.remove(0);
338       for (LatLonPoint next : tmp) {
339         LatLonPoint A = base;
340         LatLonPoint B = prev;
341         LatLonPoint C = next;
342         double a = B.distance(C);
343         double b = C.distance(A);
344         double c = A.distance(B);
345
346         if (a >= Math.ulp(a) && b >= Math.ulp(b) && c >= Math.ulp
                (c)) {
347           double angleA = Math.acos(
```

```
348             (Math.cos(a) - Math.cos(b) * Math.cos(c)) /
349             (Math.sin(b) * Math.sin(c)));
350         double angleB = Math.acos(
351             (Math.cos(b) - Math.cos(c) * Math.cos(a)) /
352             (Math.sin(c) * Math.sin(a)));
353         double angleC = Math.acos(
354             (Math.cos(c) - Math.cos(b) * Math.cos(a)) /
355             (Math.sin(a) * Math.sin(b)));
356
357         interiorAngleSum += angleA + angleB + angleC;
358         triangleCount++;
359       } else {
360         // 1 distance is 0, the other two are equal, OR
361         // all distances are 0
362         //assert a < Math.ulp(a) : A + "," + B + "," + C;
363         //assert b < Math.ulp(b) : A + "," + B + "," + C;
364         //assert c < Math.ulp(c) : A + "," + B + "," + C;
365       }
366
367       prev = next;
368     }
369
370     // approx, should use rad at current geodetic latitude
371     double earthRad = (6378.135 + 6356.750) / 2.0;
372     double area =
373       Math.pow(earthRad, 2) *
374       (interiorAngleSum - Math.PI * triangleCount);
375     return area;
376   }
377
378   private static double getScreenPolygonArea(List<Point2D>
        sortedPath){
379     if (sortedPath.size() == 0) {
380       return 0.0;
381     }
382
383     double area = 0.0;
384
385     List<Point2D> tmp = new LinkedList<Point2D>(sortedPath);
386     Point2D prev = tmp.remove(0);
387     tmp.add(prev);
388     for (Point2D next : tmp) {
389       double x1 = prev.getX();
390       double y1 = prev.getY();
391       double x2 = next.getX();
392       double y2 = next.getY();
393       area += x1*y2 - x2*y1;
394       prev = next;
395     }
396     area *= 0.5;
397     return area;
398   }
399
400   private static Point getScreenCentroid(List<Point2D> points)
```

```
401   {
402     if (points.size() < 2) {
403       return null;
404     }
405
406     double area = getScreenPolygonArea(points);
407
408     double x = 0.0;
409     double y = 0.0;
410
411     List<Point2D> tmp = new LinkedList<Point2D>(points);
412     Point2D prev = tmp.remove(0);
413     tmp.add(prev);
414     for (Point2D next : tmp) {
415       double x1 = prev.getX();
416       double y1 = prev.getY();
417       double x2 = next.getX();
418       double y2 = next.getY();
419
420       double determinant = x1*y2 - x2*y1;
421       x += (x1+x2) * determinant;
422       y += (y1+y2) * determinant;
423
424       prev = next;
425     }
426
427     x /= (6 * area);
428     y /= (6 * area);
429
430     return new Point((int)x,(int)y);
431   }
432
433   private static List<Point2D> getWindingPath(
434       CacheEntry ce, OMGraphicList circles,
435       MapBean mapBean) {
436
437     List<Point2D> ret = new LinkedList<Point2D>();
438     if (circles.size() == 0) {
439       return ret;
440     }
441
442     Proj proj = new Mercator(ce.pointCenter, ce.scale,
443         mapBean.getWidth(), mapBean.getHeight());
444     ce.listener.resetCompletion(1);
445     mapBean.setProjection(proj);
446     ce.listener.waitForCompletion();
447
448     Iterator iter = circles.iterator();
449     OMCircle circle = (OMCircle)iter.next();
450
451     Shape s = circle.getShape();
452     assert s != null;
453     Area polygon = new Area(s);
454
```

```
455       while (iter.hasNext()) {
456         circle = (OMCircle)iter.next();
457         s = circle.getShape();
458         assert s != null;
459         Area area = new Area(s);
460         polygon.intersect(area);
461       }
462
463       PathIterator pathIter = polygon.getPathIterator(null);
464       GeneralPath genPath = new GeneralPath();
465       int theType;
466       float[] theData = new float[6];
467
468       while(!pathIter.isDone()){
469         theType = pathIter.currentSegment(theData);
470
471         switch(theType){
472         case PathIterator.SEG_MOVETO :
473           genPath.moveTo(theData[0], theData[1]);
474           break;
475         case PathIterator.SEG_LINETO :
476           genPath.lineTo(theData[0], theData[1]);
477           break;
478         case PathIterator.SEG_QUADTO :
479           genPath.quadTo(theData[0], theData[1], theData[2],
                theData[3]);
480           break;
481         case PathIterator.SEG_CUBICTO :
482           genPath.curveTo(theData[0], theData[1], theData[2],
                theData[3], theData[4], theData[5]);
483           break;
484         case PathIterator.SEG_CLOSE :
485           genPath.closePath();
486           break;
487         }//end switch
488
489         pathIter.next();
490       }
491
492       PathIterator pathIter2 = genPath.getPathIterator(null);
493       FlatteningPathIterator fpi = new FlatteningPathIterator(
            pathIter2, 0.25);
494       double[] coords = new double[6];
495       while (!fpi.isDone()) {
496         fpi.currentSegment(coords);
497         ret.add(new Point2D.Double(coords[0],  coords[1]));
498         fpi.next();
499       }
500
501       return ret;
502     }
503
504
```

```
505   private static void setApproximatedPolygon(List<LatLonPoint>
          approximatePath,
506       OMGraphicList traceItems) {
507
508     if (approximatePath.size() > 0) {
509       float[] lats_lons = new float[2*(approximatePath.size()
              +1)];
510       int i = 0;
511       for (LatLonPoint point : approximatePath) {
512         lats_lons[i++] = (float)point.getLatitude();
513         lats_lons[i++] = (float)point.getLongitude();
514       }
515       LatLonPoint firstPoint = approximatePath.get(0);
516       lats_lons[i++] = (float)firstPoint.getLatitude();
517       lats_lons[i++] = (float)firstPoint.getLongitude();
518       OMPoly omPolygon = new OMAlphaPoly(lats_lons, OMGraphic.
              DECIMAL_DEGREES,
519           OMGraphic.LINETYPE_STRAIGHT, 0.1f);
520       omPolygon.setLinePaint(Color.black);
521       omPolygon.setFillPaint(Color.red);
522       traceItems.add(omPolygon);
523     }
524   }
525
526
527   private static void setBorders(CacheEntry ce, List<
          Intersection> inters,
528       OMGraphicList traceItems, MapBean mapBean) {
529     double northMost = Double.MIN_VALUE;
530     double southMost = Double.MAX_VALUE;
531     double westMost = Double.MAX_VALUE;
532     double eastMost = Double.MIN_VALUE;
533
534     if (inters.size() == 0) {
535       northMost = 90.0;
536       southMost = -90.0;
537       westMost = -180.0;
538       eastMost = 180.0;
539     } else {
540       for (Intersection inter : inters) {
541         double longitude = inter.point.getY();
542         double latitude = inter.point.getX();
543
544         if (latitude > northMost) {
545           northMost = latitude;
546         }
547         if (latitude < southMost) {
548           southMost = latitude;
549         }
550         if (longitude < westMost) {
551           westMost = longitude;
552         }
553         if (longitude > eastMost) {
554           eastMost = longitude;
```

```
555           }
556         }
557       }
558
559       ce.pointUpperLeft = new LatLonPoint(northMost, westMost);
560       ce.pointLowerRight = new LatLonPoint(southMost, eastMost);
561
562       Projection prevProj = mapBean.getProjection();
563       Point pUL = prevProj.forward(ce.pointUpperLeft);
564       Point pLR = prevProj.forward(ce.pointLowerRight);
565       ce.origScale = prevProj.getScale();
566       ce.scale = 2f * prevProj.getScale(
567           ce.pointUpperLeft, ce.pointLowerRight, pUL, pLR);
568
569       ce.pointCenter = new LatLonPoint(
570           northMost - (northMost - southMost) / 2,
571           westMost  + (eastMost - westMost) / 2);
572     }
573
574     private static void setBorderInfo(
575         CacheEntry ce, OMGraphicList traceItems) {
576
577       if (ce.centroid != null) {
578         OMPoint p = new OMPoint(
579             ce.centroid.getLatitude(),
580             ce.centroid.getLongitude(),
581             5);
582         p.setFillPaint(Color.pink);
583         traceItems.add(p);
584       }
585     }
586
587
588     private static void setPolygon2D(List<Intersection> inters,
589         OMGraphicList traceItems) {
590
591       if (inters.size() > 0) {
592         float[] lats_lons = new float[2*(inters.size()+1)];
593         int i = 0;
594         for (Intersection inter : inters) {
595           lats_lons[i++] = (float)inter.point.getX();
596           lats_lons[i++] = (float)inter.point.getY();
597         }
598         Intersection firstInter = inters.get(0);
599         lats_lons[i++] = (float)firstInter.point.getX();
600         lats_lons[i++] = (float)firstInter.point.getY();
601         OMPoly omPolygon = new OMAlphaPoly(lats_lons, OMGraphic.
              DECIMAL_DEGREES,
602             OMGraphic.LINETYPE_STRAIGHT, 0.4f);
603         omPolygon.setLinePaint(Color.black);
604         omPolygon.setFillPaint(Color.red);
605         traceItems.add(omPolygon);
606       }
607     }
```

```
608
609
610     private static void setCircles(List<Circle> circles,
611         OMGraphicList traceItems) {
612       for (Circle circle : circles) {
613         Point2D p = circle.origo;
614         LatLonPoint center = new LatLonPoint(p.getX(), p.getY());
615         OMCircle omCircle = new OMAlphaCircle(
616             center, (float)circle.rad, Length.DECIMAL_DEGREE, 0,
                    0.2f);
617         omCircle.setLinePaint(Color.black);
618         omCircle.setFillPaint(Color.black);
619         traceItems.add(omCircle);
620       }
621     }
622
623
624     private static class LayerListener implements
            LayerStatusListener {
625
626       private final Object completionMutex = new Object();
627       private final Map<Layer,Integer> completionMap =
628         new HashMap<Layer,Integer>();
629
630
631       private void addLayer(Layer layer, int remainingCount) {
632         synchronized (completionMutex) {
633           layer.addLayerStatusListener(this);
634           completionMap.put(layer, remainingCount);
635         }
636       }
637
638       public void updateLayerStatus(LayerStatusEvent evt) {
639         synchronized (completionMutex) {
640           switch (evt.getStatus()) {
641           case LayerStatusEvent.DISTRESS:
642             break;
643           case LayerStatusEvent.FINISH_WORKING:
644             completionMap.put(evt.getLayer(),
645                 completionMap.get(evt.getLayer()) - 1);
646             completionMutex.notifyAll();
647             break;
648           case LayerStatusEvent.START_WORKING:
649             break;
650           case LayerStatusEvent.STATUS_UPDATE:
651             break;
652           }
653         }
654       }
655
656
657       private void resetCompletion(int remainingCount) {
658         synchronized (completionMutex) {
659           for (Layer layer : completionMap.keySet()) {
```

```
660             completionMap.put(layer, remainingCount);
661           }
662         }
663       }
664
665       private void waitForCompletion() {
666         synchronized (completionMutex) {
667           while (true) {
668             boolean allDone = true;
669             for (Integer remaining: completionMap.values()) {
670               if (remaining > 0) {
671                 allDone = false;
672                 break;
673               }
674             }
675             if (allDone) {
676               break;
677             }
678             try {
679               long startTime = System.currentTimeMillis();
680               completionMutex.wait(5000);
681               long endTime = System.currentTimeMillis();
682               if (endTime - startTime > 4500) {
683                 break;
684               }
685             } catch (InterruptedException e) {
686               throw new RuntimeException(e);
687             }
688           }
689         }
690       }
691     }
692
693   }//class
```

C.6.7 IntersectionInfo

```
1  package kripos.geo.openmap;
2
3  /**
4   * Information about the intersection represented in a map.
5   *
6   * @author oysteine
7   */
8  public class IntersectionInfo {
9
10   /**
11    * Calculated area of intersection polygon.
12    */
13   public final double polygonArea;
14
15   /**
16    * Whether {@link #centroidLatitude} and {@link #
          centroidLongitude}
17    * contains legal values.
18    */
19   public final boolean centroidAvailable;
20
21   /**
22    * Latitude (north of Equator) of polygon centroid.
23    */
24   public final double centroidLatitude;
25
26   /**
27    * Longitude (east of Greenwich) of polygon centroid.
28    */
29   public final double centroidLongitude;
30
31   /**
32    * Pixel size of image.
33    */
34   public final int imageWidth;
35
36   /**
37    * Pixel size of image.
38    */
39   public final int imageHeight;
40
41
42   IntersectionInfo(double polygonArea, int imageWidth, int
         imageHeight) {
43     this.polygonArea = polygonArea;
44     this.centroidAvailable = false;
45     this.centroidLatitude = Double.NaN;
46     this.centroidLongitude = Double.NaN;
47
48     this.imageWidth = imageWidth;
49     this.imageHeight = imageHeight;
50   }
```

```
51
52    IntersectionInfo(double polygonArea,
53        double centroidLatitude, double centroidLongitude,
54        int imageWidth, int imageHeight) {
55      this.polygonArea = polygonArea;
56      this.centroidAvailable = true;
57      this.centroidLatitude = centroidLatitude;
58      this.centroidLongitude = centroidLongitude;
59
60      this.imageWidth = imageWidth;
61      this.imageHeight = imageHeight;
62    }
63
64  }//class
```

C.6.8 Alpha

```
1  package kripos.geo.openmap;
2
3  /**
4   * Implemented by OMGraphic subclasses that support
5   * semi-transparent fill.
6   *
7   * @author oysteine
8   * @version 1.0
9   */
10 public interface Alpha {
11
12   /**
13    * @param alpha opacity in [0.0, 1.0]
14    */
15   public void setAlpha(float alpha);
16 }
```

C.6.9 OMAlphaCircle

```
1  package kripos.geo.openmap;
2
3  import java.awt.AlphaComposite;
4  import java.awt.Composite;
5  import java.awt.Graphics;
6  import java.awt.Graphics2D;
7
8  import com.bbn.openmap.LatLonPoint;
9  import com.bbn.openmap.omGraphics.OMCircle;
10 import com.bbn.openmap.proj.Length;
11
12 /**
13  * An {@link OMCircle} implementation that supports
       transparency.
14  *
15  * Only the directly used constructors are implemented
16  *
17  * @author oysteine
18  */
19 public class OMAlphaCircle extends OMCircle implements Alpha {
20   private static final long serialVersionUID =
         9141255017478768485L;
21   private Composite composite;
22
23
24   /**
25    * Create an OMCircle with a lat/lon center and a physical
26    * distance radius. Rendertype is RENDERTYPE_LATLON.
27    *
28    * @param center LatLon center of circle
29    * @param radius distance
30    * @param units com.bbn.openmap.proj.Length object specifying
31    *        units for distance.
32    * @param nverts number of vertices for the poly-circle(if &
         lt; 3,
33    *        value is generated internally)
34    * @param alpha opacity in [0.0, 1.0]
35    */
36   public OMAlphaCircle(LatLonPoint center, float radius, Length
       units,
37       int nverts, float alpha) {
38     super(center, radius, units, nverts);
39     composite = AlphaComposite.getInstance(AlphaComposite.
         SRC_OVER, alpha);
40   }
41
42
43   /**
44    * Set the alpha color
45    *
46    * @param the color value to set
47    */
```

```
48    public void setAlpha(float alpha) {
49      composite = AlphaComposite.getInstance(AlphaComposite.
            SRC_OVER, alpha);
50    }
51
52
53    /**
54     * Overriding the fill method to set alpha before filling and
            clearing it after.
55     * @param g the <code>Graphics</code> instance to use
56     */
57    @Override
58    public void fill(Graphics g) {
59      Graphics2D g2 = (Graphics2D)g;
60      Composite orig = g2.getComposite();
61      g2.setComposite(composite);
62      super.fill(g);
63      g2.setComposite(orig);
64    }
65
66  }//class
```

C.6.10 OMAlphaPoly

```
1  package kripos.geo.openmap;
2
3  import java.awt.AlphaComposite;
4  import java.awt.Composite;
5  import java.awt.Graphics;
6  import java.awt.Graphics2D;
7  import java.awt.Paint;
8
9  import com.bbn.openmap.omGraphics.OMPoly;
10
11 /**
12  * An {@link OMPoly} implementation that supports transparency.
13  *
14  * Only the directly used constructors are implemented
15  *
16  * @author oysteine
17  */
18 public class OMAlphaPoly extends OMPoly implements Alpha {
19   private static final long serialVersionUID =
         5975120419260040 97L;
20   private Composite composite;
21   private Composite orig;
22
23
24   /**
25    *
26    * @param llPoints latitude, longitude, latitude, longitude,
          ...
27    * @param units
28    * @param lType
29    * @param alpha opacity in [0.0, 1.0]
30    */
31   public OMAlphaPoly(float[] llPoints, int units, int lType,
         float alpha) {
32     super(llPoints, units, lType);
33     composite = AlphaComposite.getInstance(AlphaComposite.
           SRC_OVER, alpha);
34   }
35
36
37   /**
38    * Set the alpha color
39    *
40    * @param the color value to set
41    */
42   public void setAlpha(float alpha) {
43     composite = AlphaComposite.getInstance(AlphaComposite.
           SRC_OVER, alpha);
44   }
45
46
47   @Override
```

```
48    public void render(Graphics g) {
49      // just to make sure we reset composite after
50      Graphics2D g2 = (Graphics2D)g;
51      orig = g2.getComposite();
52      super.render(g);
53      g2.setComposite(orig);
54    }
55
56
57    @Override
58    public void setGraphicsForFill(Graphics g) {
59      ((Graphics2D)g).setComposite(composite);
60      super.setGraphicsForFill(g);
61    }
62
63
64    @Override
65    public void setGraphicsColor(Graphics g, Paint paint) {
66      ((Graphics2D)g).setComposite(composite);
67      super.setGraphicsColor(g, paint);
68    }
69
70  }//class
```

C.6.11 Circle

```
1  //legacy code. only works in 2D. originally meant used with a
       flat map based on UTM coordinates
2  //turned out to be not accurate enough when spanning multipe
       UTM zones
3
4  //only used for setting zoom level in current implementation
5
6  package kripos.math.circle;
7
8  import java.awt.geom.Point2D;
9  import java.io.ByteArrayOutputStream;
10 import java.io.PrintStream;
11 import java.util.LinkedList;
12 import java.util.List;
13 import java.util.Locale;
14
15 /**
16  * Representation of a circle, with methods to calculate
        numbers relative to
17  * another circle.
18  *
19  * @author oysteine
20  *
21  */
22 public class Circle {
23
24   /**
25    * Center of circle;
26    */
27   public final Point2D origo;
28   /**
29    * Radius of circle
30    */
31   public double rad;
32
33   protected int intersectionCount = 0;
34
35
36   /**
37    *
38    * @param x    cartesian x coord
39    * @param y    cartesian y coord
40    * @param rad  length of radius
41    */
42   public Circle(double x, double y, double rad) {
43     this.origo = new Point2D.Double(x, y);
44     this.rad = rad;
45   }
46
47
48   /**
49    *
```

```
50      * @param angle radians from x axis
51      * @return point on circle circumference
52      */
53     public Point2D getPoint(double angle) {
54       double x = origo.getX() + rad * Math.cos(angle);
55       double y = origo.getY() + rad * Math.sin(angle);
56       return new Point2D.Double(x, y);
57     }
58
59
60     /**
61      * @param inter target: a intersection on this circle
62      * @return the angle between the x axis and the line from
63      *           this circle's origo point to <code>inter</code>'s
            point
64      */
65     double getAngle(Intersection inter) {
66       assert this == inter.c1 || this == inter.c2;
67       double opposite = inter.point.getY() - origo.getY(); //
            opposite
68       double hyp = rad;
69       if (inter.isAbove(origo)) {
70         if (inter.isRightOf(origo)) {
71           return Math.asin(opposite / hyp);
72         } else {
73           return Math.PI - Math.asin(opposite / hyp);
74         }
75       } else {
76         if (inter.isRightOf(origo)) {
77           return 2 * Math.PI + Math.asin(opposite / hyp);
78         } else {
79           return Math.PI - Math.asin(opposite / hyp);
80         }
81       }
82     }
83
84
85     /**
86      *
87      * @param inter
88      * @return <code>true</code> iff <code>inter</code> is
            neither inside
89      *           this circle's area nor on its circumference
90      */
91     boolean isOutside(Intersection inter) {
92       return !inter.belongsTo(this) &&
93       origo.distance(inter.point) > rad;
94     }
95
96
97     /**
98      *
99      * @param other
```

```
100     * @return <code>true</code> iff <code>c</code> is completely
            within
101     *          this circle's area or on its circumference
102     */
103    boolean isWithin(Circle other) {
104      if (other == this) {
105        return true;
106      }
107
108      double dist = origo.distance(other.origo);
109      return dist + other.rad <= rad;
110    }
111
112
113    /**
114     * @param other not <code>null</code>
115     * @return intersection points, or <code>null</code> if and
            only f circles do not intersect
116     *  at <i>two</i> points
117     */
118    List<Intersection> getIntersections(Circle other) {
119      assert other != null;
120
121      double maxDist = rad + other.rad;
122      double dist = origo.distance(other.origo);
123      if (dist >= maxDist) {
124        return null;
125      }
126      if (dist + Math.min(rad,other.rad) < Math.max(rad,other.rad
            )) {
127        return null;
128      }
129
130      double x1 = origo.getX();
131      double y1 = origo.getY();
132      double r1 = rad;
133      double x2 = other.origo.getX();
134      double y2 = other.origo.getY();
135      double r2 = other.rad;
136
137      double d = Math.sqrt(Math.pow(x2-x1,2) + Math.pow(y2-y1,2))
            ;
138
139      double ixPart1 = (x2+x1) / 2 + (x2-x1) * (r1*r1-r2*r2) /
            (2*d*d);
140      double ixPart2 = ( (y2-y1) / (2*d*d) ) *
141      Math.sqrt((Math.pow(r1+r2,2)-d*d) * (d*d-Math.pow(r2-r1,2))
            ) ;
142
143      double iyPart1 = (y2+y1) / 2 + (y2-y1) * (r1*r1-r2*r2) /
            (2*d*d);
144      double iyPart2 = ( (x2-x1) / (2*d*d) ) *
145      Math.sqrt((Math.pow(r1+r2,2)-d*d) * (d*d-Math.pow(r2-r1,2))
            );
```

```
146
147       Point2D first = new Point2D.Double(ixPart1+ixPart2, iyPart1
              -iyPart2);
148       Point2D second = new Point2D.Double(ixPart1-ixPart2,
              iyPart1+iyPart2);
149
150       List<Intersection> inters = new LinkedList<Intersection>();
151       inters.add(new Intersection(first, this, other));
152       inters.add(new Intersection(second, this, other));
153       return inters;
154     }
155
156
157     /**
158      * @param x
159      * @param y
160      * @return distance from the given point to circumference
161      */
162     public double getMargin(double x, double y) {
163       double margin =
164         Math.pow(x - origo.getX(), 2) +
165         Math.pow(y - origo.getY(), 2) -
166         Math.pow(rad, 2);
167       return margin;
168     }
169
170
171     @Override
172     public String toString() {
173       ByteArrayOutputStream tmp = new ByteArrayOutputStream();
174       PrintStream stream = new PrintStream(tmp);
175       stream.printf(Locale.US, "Circle(%.2f,␣%.2f,␣%.2f)",
176           origo.getX(), origo.getY(), rad);
177       return tmp.toString();
178     }
179
180   }//class
```

C.6.12 Intersection

```
1  //legacy code. only works in 2D. originally meant used with a
       flat map based on UTM coordinates
2  //turned out to be not accurate enough when spanning multipe
       UTM zones
3
4  //only used for setting zoom level in current implementation
5
6  package kripos.math.circle;
7
8  import java.awt.geom.Point2D;
9  import java.io.ByteArrayOutputStream;
10 import java.io.PrintStream;
11 import java.util.Locale;
12
13 /**
14  * Representing one intersection between two circles.
15  *
16  * @author oysteine
17  */
18 public class Intersection {
19   /** Where the two circles cross. */
20   public final Point2D point;
21   final Circle c1;
22   final Circle c2;
23
24
25   /**
26    *
27    * @param point    the intersection
28    * @param circle1  one of the participating circles
29    * @param circle2  the other participating circle
30    */
31   public Intersection(Point2D point, Circle circle1, Circle
         circle2) {
32     this.point = point;
33     this.c1 = circle1;
34     this.c2 = circle2;
35   }
36
37
38   boolean isLeftOf(Intersection c) {
39     return isLeftOf(c.point);
40   }
41   boolean isLeftOf(Point2D p) {
42     return point.getX() < p.getX();
43   }
44
45   boolean isRightOf(Intersection c) {
46     return isRightOf(c.point);
47   }
48   boolean isRightOf(Point2D p) {
49     return point.getX() > p.getX();
```

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

```
101
102
103     @Override
104     public String toString() {
105       ByteArrayOutputStream tmp = new ByteArrayOutputStream();
106       PrintStream stream = new PrintStream(tmp);
107       stream.printf(Locale.US, "p(x=%.2f,␣y=%.2f),␣%s,␣%s",
108           point.getX(), point.getY(), c1.toString(), c2.toString
                ());
109       return tmp.toString();
110     }
111
112   }//class
```

C.6.13 Intersector

```
1  //legacy code. only works in 2D. originally meant used with a
       flat map based on UTM coordinates
2  //turned out to be not accurate enough when spanning multipe
       UTM zones
3
4  //only used for setting zoom level in current implementation
5
6  package kripos.math.circle;
7
8  import java.awt.geom.Point2D;
9  import java.util.Arrays;
10 import java.util.Iterator;
11 import java.util.LinkedList;
12 import java.util.List;
13
14 /**
15  * Calculate intersections from many circles.
16  * @author oysteine
17  *
18  */
19 public class Intersector {
20
21   /**
22    *
23    * @param inters
24    * @return length of circumference of convex hull created by
           <code>inters</code>
25    */
26   static public double getPolygonLength(List<Intersection>
         inters) {
27     double length = 0.0;
28
29     if (inters.size() == 0) {
30     } else if (inters.size() == 1) {
31       // circumference of single circle
32       Intersection i = inters.get(0);
33       assert i.c1 == i.c2;
34       return 0.0;
35       //return 2.0 * Math.PI * i.c1.rad;
36     } else {
37       List<Intersection> tmp = new LinkedList<Intersection>(
             inters);
38       Intersection prev = tmp.remove(0);
39       tmp.add(prev);
40       for (Intersection next : tmp) {
41         length += prev.point.distance(next.point);
42         prev = next;
43       }
44     }
45     return length;
46   }
47
```

```
48
49    /**
50     *
51     * @param inters
52     * @return area of convex hull created by <code>inters</code>
53     */
54    static public double getPolygonArea(List<Intersection> inters
          ) {
55      if (inters.size() == 0) {
56        return 0.0;
57      }
58
59      double area = 0.0;
60
61      List<Intersection> tmp = new LinkedList<Intersection>(
            inters);
62      Intersection prev = tmp.remove(0);
63      tmp.add(prev);
64      for (Intersection next : tmp) {
65        double x1 = prev.point.getX();
66        double y1 = prev.point.getY();
67        double x2 = next.point.getX();
68        double y2 = next.point.getY();
69        area += x1*y2 - x2*y1;
70        prev = next;
71      }
72      area *= 0.5;
73      return area;
74    }
75
76    /**
77     * @param inters convex hull
78     * @return centroid of polygon or <code>null</code> if
            invalid polygon
79     */
80    static public Point2D getCentroid(List<Intersection> inters)
81    {
82      if (inters.size() < 2) {
83        return null;
84      }
85
86      double area = getPolygonArea(inters);
87
88      double x = 0.0;
89      double y = 0.0;
90
91
92      List<Intersection> tmp = new LinkedList<Intersection>(
            inters);
93      Intersection prev = tmp.remove(0);
94      tmp.add(prev);
95      for (Intersection next : tmp) {
96        double x1 = prev.point.getX();
97        double y1 = prev.point.getY();
```

```
98        double x2 = next.point.getX();
99        double y2 = next.point.getY();
100
101       double determinant = x1*y2 - x2*y1;
102       x += (x1+x2) * determinant;
103       y += (y1+y2) * determinant;
104
105       prev = next;
106     }
107
108     x /= (6 * area);
109     y /= (6 * area);
110
111     return new Point2D.Double(x,y);
112   }
113
114
115   /**
116    *
117    * @param circles
118    * @return exact size of area covered by all circles
119    */
120   static public double getExactArea(List<Circle> circles) {
121     if (circles.size() == 1) {
122       Circle c = circles.get(0);
123       return Math.PI * c.rad * c.rad;
124     }
125
126     List<Intersection> inters = getIntersections(circles);
127     if (inters.size() == 0) {
128       return 0.0;
129     }
130
131     double area = getPolygonArea(inters);
132
133     List<Intersection> tmp = new LinkedList<Intersection>(
          inters);
134     Intersection prev = tmp.remove(0);
135     tmp.add(prev);
136     for (Intersection next : tmp) {
137       ArcInfo arc = getArc(prev, next);
138
139       double asize = arc.a2 - arc.a1;
140       double sliceArea = asize / 2 * Math.pow(arc.c.rad, 2);
141       double triangleArea = Math.pow(arc.c.rad, 2) *
142       Math.cos(asize/2) * Math.sin(asize/2);
143
144       area += sliceArea - triangleArea;
145
146       prev = next;
147     }
148
149     return area;
150   }
```

```
151
152
153     /**
154      *
155      * @param inters
156      * @param distance  max distance between each point
157      * @return all original <code>inters</code> point plus points
158      *          located on the circumferences on the circles,
159      *          with maximum <code>distance</code> length between
160      *          two adjacent points
161      */
162     public static List<Intersection> getMorePoints(
163         List<Intersection> inters, double distance) {
164       List<Intersection> populated = new LinkedList<Intersection
              >();
165       if (inters.size() == 0) {
166         return populated;
167       }
168
169       //System.out.println("GETTING MORE POINTS!");
170
171       List<Intersection> tmp = new LinkedList<Intersection>(
              inters);
172       Intersection prev = tmp.remove(0);
173       tmp.add(prev);
174       for (Intersection next : tmp) {
175         populated.add(prev);
176
177         ArcInfo arc = getArc(prev, next);
178
179         //System.out.printf(arc.c + ": a1=%.2f, a2=%.2f%n", arc.
                a1, arc.a2);
180         double asize = arc.a2 - arc.a1;
181         double arclen = arc.c.rad * asize;
182         double adelta = asize / (arclen / distance);
183         for (double anew = arc.a1 + adelta; anew < arc.a2; anew
                += adelta) {
184           //System.out.printf("  adding %.2f%n", anew);
185           Point2D p = arc.c.getPoint(anew);
186           populated.add(new Intersection(p, arc.c, arc.c));
187         }
188
189
190         prev = next;
191       }
192
193       return populated;
194     }
195
196
197     private static class ArcInfo {
198       private final Circle c;
199       private final double a1;
200       private final double a2;
```

```
201
202       private ArcInfo(Circle c, double a1, double a2) {
203         this.c = c;
204         this.a1 = a1;
205         this.a2 = a2;
206       }
207     }
208
209     private static ArcInfo getArc(Intersection prev, Intersection
          next) {
210       // find potential circles
211       Circle c = prev.c1;
212       Circle cc = null;
213       if (!next.belongsTo(c)) {
214         c = prev.c2;
215       } else {
216         cc = prev.c2;
217         if (!next.belongsTo(cc)) {
218           cc = null;
219         }
220       }
221
222       // find degrees and wanted circle
223
224       double a1 = c.getAngle(prev);
225       double a2 = c.getAngle(next);
226       if (a2 <= a1) { // equals to support a single circle with a
            single point
227         a2 += 2 * Math.PI;
228       }
229
230       if (cc != null && cc != c &&
231           (a2 - a1 > Math.PI || c.rad < cc.rad)) {
232         double b1 = cc.getAngle(prev);
233         double b2 = cc.getAngle(next);
234         if (b2 < b1) {
235           b2 += 2 * Math.PI;
236         }
237
238         if (b2 - b1 < Math.PI ||
239             cc.rad < c.rad) {
240           c = cc;
241           a1 = b1;
242           a2 = b2;
243         }
244       }
245
246       return new ArcInfo(c, a1, a2);
247     }
248
249
250
251     /**
252      *
```

```
253     * @param circles
254     * @return all points where circles cross to make up
255     *          the area all circles overlap
256     */
257    public static List<Intersection> getIntersections(List<Circle
           > circles) {
258
259      List<Intersection> inters = new LinkedList<Intersection>();
260
261      if (circles.size() == 0) {
262        return inters;
263      }
264
265      // set intersections at each circle
266      List<Circle> sources = new LinkedList<Circle>(circles);
267      List<Circle> targets = new LinkedList<Circle>();
268
269      Circle firstSource = sources.remove(0);
270      targets.add(firstSource);
271      // add dummy intersection
272      inters.add(new Intersection(
273          firstSource.getPoint(0), firstSource, firstSource));
274      firstSource.intersectionCount += 2;
275
276      while (!sources.isEmpty()) {
277        Circle source = sources.remove(0);
278
279        // remove previous intersections that fall outside new
               source
280        Iterator<Intersection> iterator = inters.iterator();
281        while (iterator.hasNext()) {
282          Intersection inter = iterator.next();
283          if (source.isOutside(inter)) {
284            iterator.remove();
285            inter.c1.intersectionCount--;
286            inter.c2.intersectionCount--;
287          }
288        }
289
290        // add new intersections from source
291        for (Circle target : targets) {
292          List<Intersection> newInters = target.getIntersections(
                   source);
293
294          if (newInters != null) {
295            for (Intersection newInter : newInters) {
296              boolean valid = true;
297              for (Circle target2 : targets) {
298                if (target2.isOutside(newInter)) {
299                  valid = false;
300                  break;
301                }
302              }
303              if (valid) {
```

```
304                 inters.add(newInter);
305                 newInter.c1.intersectionCount++;
306                 newInter.c2.intersectionCount++;
307               }
308             }
309           }
310         }
311
312         if (inters.size() == 0) {
313           // add dummy intersection if source is within all
                targets
314           boolean withinAll = true;
315           for (Circle target : targets) {
316             if (!target.isWithin(source)) {
317               withinAll = false;
318               break;
319             }
320           }
321
322           // add dummy intersection
323           if (withinAll) {
324             inters.add(new Intersection(
325                 source.getPoint(0), source, source));
326             source.intersectionCount += 2;
327           }
328         } else if (inters.size() > 1) {
329           // remove dummy intersection
330           Intersection test = inters.get(0);
331           if (test.c1 == test.c2) {
332             inters.remove(0);
333             test.c1.intersectionCount -= 2;
334           }
335         }
336
337         // add source as target
338         targets.add(source);
339
340         // remove targets without any intersections
341         Iterator<Circle> citer = targets.iterator();
342         while (citer.hasNext()) {
343           Circle target = citer.next();
344           assert target.intersectionCount >= 0 :
345             target.toString() + ",␣intersCount=" + target.
                  intersectionCount;
346           if (target.intersectionCount == 0) {
347             citer.remove();
348           }
349         }
350       }
351
352       if (inters.size() == 0) {
353         return inters;
354       }
355
```

```
356
357
358       // find lowest point to start convex polygon creation
359       Intersection bottom = inters.get(0);
360       for (Intersection inter : inters) {
361         if (inter.isBelow(bottom)) {
362           bottom = inter;
363         }
364       }
365       inters.remove(bottom);
366
367       // find angles from bottom to rest of intersections
368       AngledIntersection[] angles = new AngledIntersection[inters
              .size()];
369       int i = 0;
370       for (Intersection inter : inters) {
371         angles[i++] = new AngledIntersection(
372             inter,
373             bottom.getAngle(inter));
374       }
375
376       // sort angles and add
377       List<Intersection> sorted = new LinkedList<Intersection>();
378       sorted.add(bottom);
379       Arrays.sort(angles);
380       for (AngledIntersection tmp : angles) {
381         sorted.add(tmp.inter);
382       }
383
384       return sorted;
385     }
386
387
388     private static class AngledIntersection implements Comparable
              <AngledIntersection> {
389       private final Intersection inter;
390       private final double angle;
391       private AngledIntersection(Intersection inter, double angle
              ) {
392         this.inter = inter;
393         this.angle = angle;
394       }
395
396       public int compareTo(AngledIntersection other) {
397         if (angle < other.angle) {
398           return -1;
399         } else if (angle > other.angle) {
400           return 1;
401         } else {
402           // TODO: compare distance
403           return 0;
404         }
405       }
406     }
```

```
407 }
```

C.7 DB Classes

C.7.1 DBCreator

```
1  package kripos.tools;
2
3  import java.net.InetAddress;
4  import java.net.UnknownHostException;
5  import java.sql.Connection;
6  import java.sql.DatabaseMetaData;
7  import java.sql.DriverManager;
8  import java.sql.ResultSet;
9  import java.sql.SQLException;
10 import java.sql.Statement;
11 import java.util.ArrayList;
12
13 import kripos.geo.Landmark;
14
15 /**
16  * Creates and fills the database used by agents for
17  * storing geolocation information
18  * Any existing database and content will be dropped!
19  *
20  * @author oysteine
21  * @version
22  *
23  */
24 public class DBCreator {
25   private String dbmsPath;//TODO not used yet
26   private String user;//TODO not used yet
27   private String pw;//TODO not used yet
28   private String myName;
29
30   /**
31    * Creates the database tables relative to the landmark host
32    */
33   public DBCreator() {
34     InetAddress iadr;
35     try {
36       iadr = InetAddress.getLocalHost();
37       myName = iadr.getCanonicalHostName();
38     } catch (UnknownHostException e) {
39       System.out.println("Unable to get local fqdn hostname");
40       e.printStackTrace();
41     }
42   }
43
44   /**
45    * Establishes a connection to the database
46    *
47    * @return con connection to the database
48    */
```

```
49    private Connection connect() throws SQLException,
          ClassNotFoundException{
50      Connection c = null;
51      Class.forName("org.hsqldb.jdbcDriver");
52      c = DriverManager.getConnection("jdbc:hsqldb:hsql://
            localhost/xdb", "sa", "");
53      return c;
54    }
55
56    /**
57     * Creates and fills the table containing information about
          landmarks
58     *
59     * @param Database connection con
60     */
61    private void createLandmarkTable(Connection con) throws
          SQLException{
62      String landmarkTable = "CREATE␣TABLE␣LANDMARKS␣" +
63      "(NAME␣VARCHAR(32)␣NOT␣NULL,␣IPADR␣VARCHAR(39)␣NOT␣NULL,␣
            CHECKED␣TIMESTAMP,␣" +
64      "DISTANCE_KM␣DOUBLE␣NOT␣NULL,␣LATITUDE␣DOUBLE␣NOT␣NULL,␣
            LONGITUDE␣DOUBLE␣NOT␣NULL,␣MIN_RTT␣DOUBLE,␣" +
65      "AVG_RTT␣DOUBLE,␣C1␣DOUBLE,␣EPSILON␣DOUBLE,␣HASH␣VARCHAR
            (64),␣PRIMARY␣KEY(NAME,IPADR))";
66
67      Statement stmt;
68      stmt = con.createStatement();
69      //create table
70      stmt.executeUpdate(landmarkTable);
71      //fill table
72      LandmarkReader lr = new LandmarkReader();
73      ArrayList<Landmark> landmarks = lr.distance(myName);
74
75      for(int i=0;i<landmarks.size();i++){
76        Landmark l = landmarks.get(i);
77        double distance = l.getDistance();
78        double latitude = l.getGeoPosition().getLatitude();
79        double longitude = l.getGeoPosition().getLongitude();
80        String IP = l.getIP();
81        String name = l.getName();
82        String landmarkAdd = "INSERT␣INTO␣LANDMARKS␣" +
83        "VALUES␣('"+name+"','"+IP+"',null,"+distance+","+latitude
              +","+longitude+"," +
84        9999999+"," + -1+"," + -1+"," + -1+",'A')";
85
86        stmt.executeUpdate(landmarkAdd);
87      }
88    }
89
90    /**
91     * Creates the table containing information about traced
          hosts
92     *
93     * @param Database connection con
```

```
94      */
95     private void createTraceTable(Connection con) throws
           SQLException{
96       //contains traced hosts and info
97       String traceTable = "CREATE␣TABLE␣TRACED␣" +
98       "(NAME␣VARCHAR(32),␣IPADR␣VARCHAR(39),␣CHECKED␣TIMESTAMP␣
             NOT␣NULL,␣" +
99       "MIN_RTT␣DOUBLE␣NOT␣NULL,␣AVG_RTT␣DOUBLE,␣C1␣DOUBLE,␣
             EPSILON␣DOUBLE,␣HASH␣VARCHAR(64),␣PRIMARY␣KEY(IPADR))";
100
101      Statement stmt;
102      stmt = con.createStatement();
103      stmt.executeUpdate(traceTable);
104    }
105
106    /**
107     * Creates the table containing misc information,
108     *
109     * @param Database connection con
110     */
111    private void createMiscTable(Connection con)throws
           SQLException{
112      //contains misc data. when last bestline etc.
113      String miscTable = "CREATE␣TABLE␣MISC␣" +
114      "(NAME␣VARCHAR(32),␣IPADR␣VARCHAR(39),␣LAST_BESTLINE␣
             TIMESTAMP,␣BESTLINE_M␣DOUBLE,␣BESTLINE_B␣DOUBLE,␣
             LATITUDE␣DOUBLE,␣LONGITUDE␣DOUBLE)";
115
116      Statement stmt = con.createStatement();
117      //create table
118      stmt.executeUpdate(miscTable);
119
120      //fill table
121      LandmarkReader lr = new LandmarkReader();
122      Landmark l = lr.getSingleLandmark(myName);
123
124      double latitude = l.getGeoPosition().getLatitude();
125      double longitude = l.getGeoPosition().getLongitude();
126      String IP = l.getIP();
127
128      String miscAdd = "INSERT␣INTO␣MISC␣VALUES␣('"+myName+"','"+
             IP+"',null,0,0,"+latitude+","+longitude+")";
129      stmt.executeUpdate(miscAdd);
130    }
131
132    /**
133     * @param args
134     * @throws SQLException
135     */
136    public static void main(String[] args) {
137      DBCreator dbCreate = new DBCreator();
138      try{
139        Connection con = dbCreate.connect();
140        Statement stmt;
```

```
141         stmt = con.createStatement();
142         DatabaseMetaData dbmd = con.getMetaData();
143         ResultSet rs1 = dbmd.getTables(null,null,null,null);
144         //drop all existing normal tables
145         while(rs1.next()){
146           String tableName = rs1.getString("TABLE_NAME");
147           String tableType = rs1.getString("TABLE_TYPE");
148           if(tableType.equalsIgnoreCase("TABLE")){
149             stmt.execute("DROP␣TABLE␣" +tableName);
150           }
151         }
152         //create and fill tables
153         dbCreate.createLandmarkTable(con);
154         dbCreate.createTraceTable(con);
155         dbCreate.createMiscTable(con);
156         con.close();
157         System.out.println("Database␣created␣successfully!");
158         System.exit(0);
159 
160       } catch (SQLException se) {
161         System.out.println("Database␣creation␣FAILED!");
162         se.printStackTrace();
163         System.exit(1);
164       }catch (ClassNotFoundException ce) {
165         System.out.println("Database␣creation␣FAILED!");
166         ce.printStackTrace();
167         System.exit(1);
168       }
169     }
170 
171 }//class
```

C.7.2 DBStop

```
1  package kripos.tools;
2
3  import java.sql.Connection;
4  import java.sql.DriverManager;
5  import java.sql.SQLException;
6  import java.sql.Statement;
7
8  /**
9   * @author oysteine
10  *
11  */
12 public class DBStop {
13
14   public DBStop() {
15   }
16
17   public static void main(String[] args){
18     DBStop dbs = new DBStop();
19     try {
20       Connection con = dbs.connect();
21       Statement st = con.createStatement();
22       st.execute("SHUTDOWN");
23
24     } catch (SQLException e) {
25       e.printStackTrace();
26     } catch (ClassNotFoundException e) {
27       e.printStackTrace();
28     }
29   }
30
31   /**
32    * Establishes a connection to the database
33    *
34    * @return con connection to the database
35    */
36   private Connection connect() throws SQLException,
         ClassNotFoundException{
37     Connection c = null;
38     Class.forName("org.hsqldb.jdbcDriver");
39     c = DriverManager.getConnection("jdbc:hsqldb:hsql://
         localhost/xdb", "sa", ""); //FIXME
40     return c;
41   }
42
43 }
```

C.7.3 LandmarkReader

```
1  package kripos.tools;
2
3  import java.io.BufferedReader;
4  import java.io.FileNotFoundException;
5  import java.io.FileReader;
6  import java.io.IOException;
7  import java.net.InetAddress;
8  import java.util.ArrayList;
9  import kripos.geo.Landmark;
10 import com.bbn.openmap.LatLonPoint;
11 import com.bbn.openmap.proj.Length;
12 import com.bbn.openmap.proj.coords.UTMPoint;
13
14 /**
15  * Tool to read landmarkinformation from file.
16  * Includes method to calculate distance between landmarks
17  *
18  * @author oysteine
19  * @version 1.1
20  *
21  */
22 public class LandmarkReader {
23   private ArrayList<Landmark> landmarks = new ArrayList<
         Landmark>();
24
25   /**
26    * Creates a LandmarkReader and reads in landmarkinformation
          from file
27    */
28   public LandmarkReader() {
29     convert();
30   }
31
32   /**
33    * Calculates and returns the distances to all landmarks
34    * from the landmark with the name provided.
35    *
36    * @param name of landmark to compute distances from
37    * @return list of landmarks with distances.
38    */
39   public ArrayList<Landmark> distance(String landmarkName){
40     Landmark from = null;
41     for(int i=0;i<landmarks.size();i++){
42       if(landmarkName.equalsIgnoreCase(landmarks.get(i).getName
             ())){
43         from = landmarks.remove(i);
44         break;
45       }
46     }
47
48     for(int i=0; i<landmarks.size();i++){
```

```
49        double radDistance = from.getGeoPosition().distance(
              landmarks.get(i).getGeoPosition());
50        Length converter = Length.KM;
51        landmarks.get(i).setDistance(converter.fromRadians(
              radDistance));
52      }
53      return landmarks;
54    }
55
56    /**
57     * Get information about a single landmark in the form of
          Landmark object
58     *
59     * @param landmarkName the name of the
60     * @return the Landmark object for the landmark queried for
61     */
62    public Landmark getSingleLandmark(String landmarkName){
63      Landmark result = null;
64      for(int i=0;i<landmarks.size();i++){
65        if(landmarkName.equalsIgnoreCase(landmarks.get(i).getName
              ())){
66          result = landmarks.remove(i);
67          break;
68        }
69      }
70      return result;
71    }
72
73    /**
74     * Reads landmarkinformation from file
75     * | is used as field separator
76     * lines starting with # are ignored
77     */
78    private void convert(){
79      try {
80        FileReader fr = new FileReader("maalepaaler.txt");
81        BufferedReader br = new BufferedReader(fr);
82        String temp = null;
83
84        while((temp = br.readLine()) !=null){
85          if('#' == temp.charAt(0)){ //ignore lines starting with
                #
86          }
87          else{
88            int firstCutPoint = temp.indexOf('|');
89            String name = temp.substring(0, firstCutPoint);
90            int secondCutPoint = temp.indexOf('|', firstCutPoint
                  +1);
91            int UTMZone = Integer.parseInt(temp.substring(
                  firstCutPoint+1, secondCutPoint));
92            int thirdCutPoint = temp.indexOf('|', secondCutPoint
                  +1);
93            int UTMNorthing = Integer.parseInt(temp.substring(
                  secondCutPoint+1, thirdCutPoint));
```

```
94              int UTMEasting = Integer.parseInt(temp.substring(
                    thirdCutPoint+1));
95              UTMPoint utmp = new UTMPoint(UTMNorthing, UTMEasting,
                     UTMZone,'N');
96              LatLonPoint llp = utmp.toLatLonPoint();
97
98              //get the current IP-address of the hostname
99              InetAddress iadr = InetAddress.getByName(name);
100             String IP = iadr.getHostAddress();
101
102             Landmark l = new Landmark(name, llp, IP);
103             landmarks.add(l);
104           }
105         }
106       } catch (FileNotFoundException e) {
107         e.printStackTrace();
108       } catch (IOException e) {
109         e.printStackTrace();
110       }
111     }
112
113 }//class
```

C.7.4 Landmark

```
1  package kripos.geo;
2
3  import java.util.Date;
4
5  import com.bbn.openmap.LatLonPoint;
6
7  /**
8   * Contains information about a single Landmark,
9   * relative to the position of the owner of this instance.
10  *
11  * @author oysteine
12  * @version 1.0
13  */
14 public class Landmark {
15   private String myName;
16   private LatLonPoint geoPosition; //TODO
17   private double myDistance;
18   private String myIP;
19   private Date lastChecked;
20   private double minRTT = 0;
21   private double avgRTT;
22   private double C1;
23   private double epsilon;
24   private String hashedTimestamp; //TODO
25
26   /**
27    *
28    */
29   public Landmark(String IP) {
30     myIP = IP;
31   }
32
33   /**
34    *
35    */
36   public Landmark(String name, LatLonPoint llp, String IP) {
37     myName = name;
38     myIP = IP;
39     geoPosition = llp;
40   }
41
42   /**
43    *
44    */
45   public Landmark(String name, double distance, String IP) {
46     myName = name;
47     myDistance = distance;
48     myIP = IP;
49   }
50
51   /**
52    * @return the geoPosition
```

```
53      */
54     public LatLonPoint getGeoPosition() {
55       return geoPosition;
56     }
57
58     /**
59      * @param myDistance the myDistance to set
60      */
61     public void setDistance(double myDistance) {
62       this.myDistance = myDistance;
63     }
64
65     /**
66      * @return the avgRTT
67      */
68     public double getAvgRTT() {
69       return avgRTT;
70     }
71
72     /**
73      * @param avgRTT the avgRTT to set
74      */
75     public void setAvgRTT(double avgRTT) {
76       this.avgRTT = avgRTT;
77     }
78
79     /**
80      * @return the c1
81      */
82     public double getC1() {
83       return C1;
84     }
85
86     /**
87      * @param c1 the c1 to set
88      */
89     public void setC1(double c1) {
90       C1 = c1;
91     }
92
93     /**
94      * @return the epsilon
95      */
96     public double getEpsilon() {
97       return epsilon;
98     }
99
100    /**
101     * @param epsilon the epsilon to set
102     */
103    public void setEpsilon(double epsilon) {
104      this.epsilon = epsilon;
105    }
106
```

```
107     /**
108      * @return the hashedTimestamp
109      */
110     public String getHashedTimestamp() {
111       return hashedTimestamp;
112     }
113
114     /**
115      * @param hashedTimestamp the hashedTimestamp to set
116      */
117     public void setHashedTimestamp(String hashedTimestamp) {
118       this.hashedTimestamp = hashedTimestamp;
119     }
120
121     /**
122      * @return the lastChecked
123      */
124     public Date getLastChecked() {
125       return lastChecked;
126     }
127
128     /**
129      * @param lastChecked the lastChecked to set
130      */
131     public void setLastChecked(Date lastChecked) {
132       this.lastChecked = lastChecked;
133     }
134
135     /**
136      * @return the minRTT
137      */
138     public double getMinRTT() {
139       return minRTT;
140     }
141
142     /**
143      * @param minRTT the minRTT to set
144      */
145     public void setMinRTT(double minRTT) {
146       this.minRTT = minRTT;
147     }
148
149     /**
150      * @return the distance
151      */
152     public double getDistance() {
153       return myDistance;
154     }
155
156     /**
157      * @param distance the distance to set
158      */
159     public void SetDistance(double distance) {
160       myDistance = distance;
```

```
161     }
162
163     /**
164      * @return the ip
165      */
166     public String getIP() {
167       return myIP;
168     }
169
170     /**
171      * @return the name
172      */
173     public String getName() {
174       return myName;
175     }
176
177 }//class
```

C.8 Scripts used for Managing the System

C.8.1 Unidist

```
#!/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
done

printf "Copied␣current␣version␣to␣all␣unihosts"
printf "\n"
```

C.8.2 Unirun

```
#!/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/
    commons-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"
```

```
7
8  if [[ ! -e "$hostfile" ]]
9  then
10          printf "${hostfile##*/}non-existent"
11 exit 1
12 fi
13
14 for host in $(cat $hostfile)
15 do
16     ssh $uniuser@$host java -cp $classpath1: org.hsqldb.Server
           -database.0 mydb -dbname.0 xdb &
17          ssh $uniuser@$host java -cp $classpath2 jade.Boot -
               nomobility -container -host $sjef -container-name
               $host $host-Admin:kripos.geo.AdminAgent &
18     ssh $uniuser@$host java -cp $classpath2 jade.Boot -
           container -host $sjef -container-name $host $host-Admin:
           kripos.geo.AdminAgent &
19          sleep 1
20 done
21
22 printf "system␣started"
```

C.8.3 Unistop

```
1  #!/bin/bash
2  hostfile=~/diplom/shell/unihosts
3  uniuser="oysteine"
4  classpath1="~/hsqldb/lib/hsqldb.jar:~/geolocate.jar"
5
6  if [[ ! -e "$hostfile" ]]
7  then
8      printf "${hostfile##*/}non-existent"
9  exit 1
10 fi
11
12 for host in $(cat $hostfile)
13 do
14     ssh $uniuser@$host java -cp $classpath1: kripos.tools.
           DBStop
15     ssh $uniuser@$host killall java
16 done
17
18 printf "system␣stopped"
```

C.8.4 Unikill

```
1  #!/bin/bash
2  hostfile=~/diplom/shell/unihosts
3  uniuser="oysteine"
4  classpath1="~/hsqldb/lib/hsqldb.jar:~/geolocate.jar"
5
6
7  if [[ ! -e "$hostfile" ]]
```

```
8  then
9          printf "${hostfile##*/}non-existent"
10 exit 1
11 fi
12
13 for host in $(cat $hostfile)
14 do
15         ssh $uniuser@$host java -cp $classpath1: kripos.tools.
               DBStop
16         ssh $uniuser@$host killall -9 java
17 done
```

C.8.5 Unicdb

```
1  #!/bin/bash
2  hostfile=~/diplom/shell/unihosts
3  uniuser="oysteine"
4  classpath1=".:/home/oysteine/hsqldb/lib/hsqldb.jar:/home/
       oysteine/openmap/lib/openmap.jar"
5
6  if [[ ! -e "$hostfile" ]]
7  then
8     printf "${hostfile##*/}non-existent"
9  exit 1
10 fi
11
12 for host in $(cat $hostfile)
13 do
14    printf "$host **************"
15    printf "\n"
16    ssh $uniuser@$host java -cp $classpath1: org.hsqldb.Server -
          database.0 mydb -dbname.0 xdb &
17    sleep 2
18    ssh $uniuser@$host java -cp $classpath1 kripos.tools.
          DBCreator &
19 done
```

C.9 JADE properties files

C.9.1 JADE-S main.conf

```
1  # ---- JADE configuration ----
2
3  # ------ Services  ------
4  services=\
5  jade.core.security.SecurityService;\
6  jade.core.security.signature.SignatureService;\
7  jade.core.security.encryption.EncryptionService;\
8  jade.core.event.NotificationService
9
10
```

```
11 # ------ Agents  ------
12 agents=kripos-rma:jade.tools.rma.rma
13
14 # ------ Security configuration ------
15
16 # ---- Permission ----
17 # Permission Policy file
18 java.security.policy=policy.txt
19
20
21 # ---- Authentication ----
22
23 # - Type of Prompt
24 jade.security.authentication.logincallback=Cmdline
25
26 # - if Cmdline, use this user/pass -
27 owner=kripos:test
28
29 # - Auth module
30 jade.security.authentication.loginmodule=Simple
31
32 # - if Simple, use this password file
33 jade.security.authentication.loginsimplecredfile=passwords.txt
34
35
36 # - JAAS configuration file -
37 java.security.auth.login.config=jaas.conf
38
39 # ---- end JADE configuration ----
```

C.9.2 jaas.conf

```
/*
 * 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

```
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

```
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 an 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 $\pm$ 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 the 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.

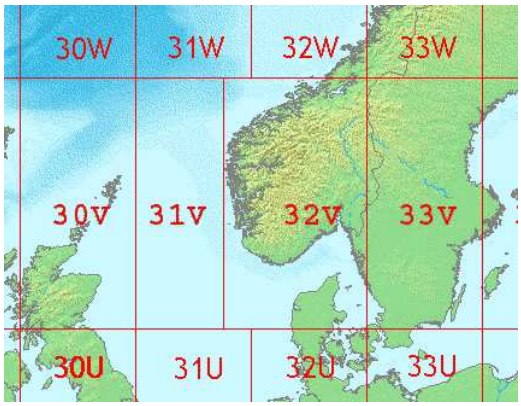


Figure D.1: The extended UTM zone 32V and the shrunk 31V.

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].