Semantic Geocast Using a Self-Organizing Infrastructure

Jörg Roth

University of Hagen
Department for Computer Science
58084 Hagen, Germany
Joerg.Roth@Fernuni-hagen.de

Abstract. Geocast mechanisms allow a sender to transmit network packets to receivers residing at a certain geographical region. Geocast forms the basis for a number of location-based services, such as announcement services, advertisement services or friend-finders. In this paper, we introduce the notion of *semantic geocast*, where a target area is specified by its *meaning*. A sender can broadcast messages to, e.g., a city centre or a specific building, without precisely knowing the physical co-ordinates. We implemented semantic geocast on top of our self-organizing *Location Server Infrastructure* (*LSI*), which reflects a location domain model especially designed to cover the needs of mobile users. As our infrastructure is self-organizing, it is flexible and easy to extend. We consider scalability and stability issues. LSI and its geocast mechanism is fully implemented and tested. Evaluations show the effectiveness of our approach.

1 Introduction

Location-based services will become increasingly popular in the future. Applications that take into account a mobile user's current location play a major role in the area of ubiquitous, pervasive and handheld computing. Many people expect a high potential of location-based services such as city guides or navigation systems for m-commerce scenarios. To support developers of location-based services we created the platform *LSI* (*Location Server Infrastructure*). LSI hides the specific mechanism to determine the mobile user's current location and provides both physical co-ordinates as well as semantic information about the current location. With LSI, mobile users can switch between satellite navigation systems such as GPS, positioning systems based on cell-phone infrastructures, or indoor positioning systems without affecting the location based-service. A developer can concentrate on the actual service function and has not to deal with positioning sensors or capturing protocols.

One powerful tool to develop location-based services is *geocast*. Like multicast, geocast transmits a network packet to a number of receivers, but in contrast to multicast, the target is a certain geographical region. Geocast is an ideal basic function for a number of location-based applications. With geocast, we can send warning announcements to a region with bad weather conditions, supermarkets can send adver-

tisement messages to all clients inside a building, and friend-finder applications can look for friends in the nearer area of a mobile user.

In this paper, we introduce the notion of *semantic geocast*: the target address is not defined by a *physical* area (specified by, e.g., a polygon), but by a *semantic* location such as "University of Hagen". Using semantic geocast, users and applications do not have to deal with raw physical co-ordinates, but can use simple location names to describe the target.

2 Related Work

The notion of geocast was introduced by Imielinski and Navas ([5], [6], and [11]). As a basic idea, geocast extends traditional networks by services to use geographical target addresses. In [11], Navas and Imielinski suggested a hierarchical network of *GeoRouters* that reflects a structure of a wireless cellular (e.g., cell-phone) network. As there is no notion of semantic locations, we can only use physically defined targets. In [5], semantic locations are partly supported, as some semantic locations (e.g., countries and cities) are represented by individual multicast addresses using multicast IP group addresses. This approach, however, was not scalable, because the number of potential multicast IP groups is far to small to cover a reasonable area such as an entire country. In addition, the multicast IP infrastructure is not prepared for a huge number of multicast members, moreover not generally available for mobile users.

Many location-based applications have been developed in the last years, which use semantic locations. Cyberguide [1], Guide [2] and the PinPoint Tourist Guide [16] offer information to tourists, taking into account their current (semantic) location. Context-aware messaging tools trigger actions according to a specific semantic location [18]. ComMotion [10] and CybreMinder [3] link locations to events, e.g. give an alarm if time is "9:00" and location is "my office". These tourist guides and messaging services use their own, hard-coded mechanisms to express semantic locations. They would heavily benefit from a general infrastructure to use semantic geocast.

Several research platforms provide a basis to develop location-based services. Cooltown [8] is a collection of location-aware applications, tools and development environments. Nexus [4] introduces so-called augmented areas to formalize location information. Augmented areas represent spatially limited areas, which may contain real as well as virtual objects. OpenLS [12] is an upcoming project and provides a high-level framework to build location-based services. All these systems could heavily benefit from a framework supporting semantic locations as well as semantic geocast services.

3 Semantic Locations and the Location Server Infrastructure

The notion of semantic locations is not new (e.g., [9], [18]), but descriptions often tend to be very abstract. Pradhan distinguishes three types of locations [14]: *physical* locations such as GPS coordinates, *geographical* locations such as "City of Hagen" and *semantic* locations such as "Jörg's office at the university". In this paper, we do

not distinguish geographical and semantic locations, but view any location other than physical as a semantic location. In the following, we first introduce a formal model for semantic locations. We then present our infrastructure LSI, which reflects this formal model.

3.1 Semantic Locations

Semantic locations are appropriate for a number of applications, sometimes in combination with physical locations. Semantic locations have some important advantages:

- Semantic locations have a meaning to the user; in contrast, physical locations usually have no meaning at all to most peoples.
- Semantic locations can easily be used as a search key for traditional databases, tables or lists. Looking up physical presentations, we need spatial databases with the ability to deal with geometric objects such as polygons.

In this section, we want to describe the concept of semantic locations more precisely. We especially want to relate semantic locations to physical locations. Let P denote the set of all physical locations. We call each coherent area $S \subseteq P$ a *semantic location* of P. We further call each set $C \subseteq 2^P$ of semantic locations, a *semantic coordinate system* of P. (2^P denotes the power set of P.) Note that we do not assume two semantic locations to be generally disjoint. A reasonable semantic coordinate system C contains semantic locations S with certain meanings, e.g.,

- locations with a political meaning: countries, states, districts, cities;
- geographical locations: continents, mountains, rivers, lakes, forests;
- mobile locations: trains, planes, cars;
- temporary locations: construction zones, fairs;
- other locations: campus, malls, city centres.

We further introduce a *name* for a semantic location. Let N be the set of all possible names. We define a function $NAME$: $C \rightarrow N$, which maps a semantic location to a string. We require names to be unique, i.e. $NAME(c_1) \neq NAME(c_2)$ for $c_1 \neq c_2$. We call a semantic location with its corresponding name a *domain*. For a domain d, $d.name$ denotes the domain name, $d.c$ the semantic location.

In principle, a semantic coordinate system C could be an arbitrary subset of 2^P that contains coherent areas. Looking at real-world scenarios, however, we usually find hierarchical structures (fig. 1), e.g., a room is inside a building, a building is in a city, a city is in a country etc.

We divide C into so-called *hierarchies*. A hierarchy contains domains with a similar meaning, e.g., domains of german cities or domains of geographical items. Each hierarchy has a *root domain* and a number of *subdomains*; each of it can in turn be divided into subdomains. We call a top node of a subhierarchy a *master* of the corresponding subdomains. We denote $m \rhd s$ for master m of subdomain s. Further $\succ$ denotes the reflexive and transitive closure of $\rhd$, i.e. $d_1 \succ d_2$ if either $d_1 = d_2$ or d_1 is a top node of a subtree which contains d_2.

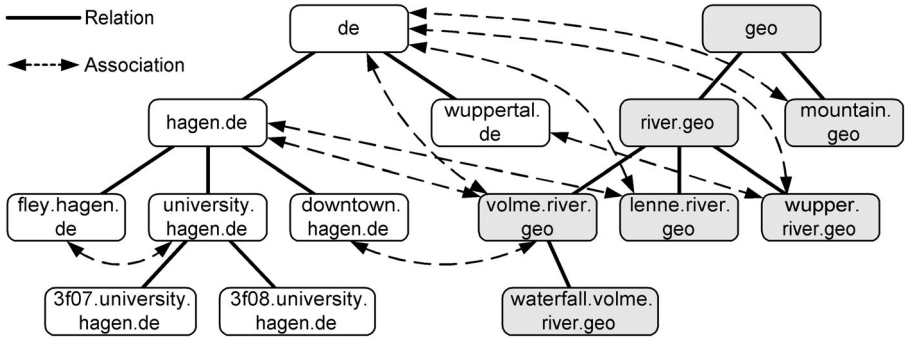

Fig. 1. A sample semantic coordination system

Fig. 1 shows two hierarchies, a `de` hierarchy (the area of Germany, white boxes) and a `geo` hierarchy (geographical entities such as rivers and mountains, grey boxes).

We call a link between a subdomain and its master *relation*. Relations carry information about containment of one domain according to another. Hierarchies are built according to three rules:

- The area of a subdomain has to be completely inside the area of its master, i.e. if $d_1 \triangleright d_2$ then $d_2.c \subset d_1.c$.
- The name of a subdomain d_2 extends the name of its master d_1 according to the rule $d_2.name$=<extension> + '.' + $d_1.name$, where <extension> can be an arbitrary string containing only letters and digits. With the help of this rule, we can effectively check if $d_1 \succ d_2$ or $d_1 \triangleright d_2$ with the help of the domain names.
- Root domain names of two hierarchies must be different.

In addition to relations, a domain can be *associated* to other domains. Two domains d_1, d_2 are associated, if they share a physical area (i.e. $d_1.c \cap d_2.c \neq \{\}$) and neither $d_1 \succ d_2$ nor $d_2 \succ d_1$. Associated domains can be in different hierarchies or in the same hierarchy. The domain `downtown.hagen.de` is associated to `volme.river.geo`, because Volme is a river which flows through the downtown of Hagen. Associations carry important information for location-based services. E.g. with the help of associations, we can discover *all* semantic locations of a specific physical location.

The number of associations can be very high for high-level domains. We reduce the amount of associations with a compression mechanism [17], which deletes associations without loosing the corresponding information. In fig. 1, e.g. `de` is not associated to `geo`, since lower-level associations carry all necessary information.

3.2 The Technical Infrastructure

Our technical infrastructure LSI reflects the domain model described above. A distributed system of so-called *location servers* (*LS*) stores location information and provides services for mobile clients and the corresponding location based services (fig 2). The infrastructure consists of three *segments*:

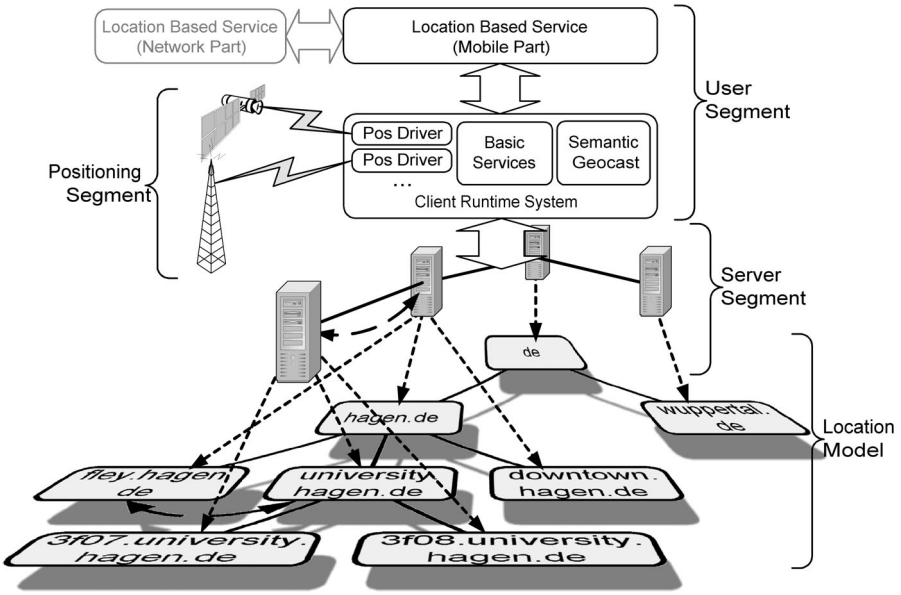

Fig. 2. The system architecture

The *positioning segment* contains the positioning systems, e.g., indoor positioning systems, satellite navigation systems or systems based on cell phone infrastructures. The *user segment* contains the mobile node with the LSI runtime system and the mobile part of the location based service. Note that our infrastructure does not cover the network part of a location-based service. It depends on the mobile part to establish a connection to a specific server and to use the service. The runtime system accesses the positioning systems through *position drivers*. With the help of drivers, we can change the positioning systems, even at runtime, without affecting the rest of the system. The client runtime system also contains the following components:

- *Basic services* provide a homogenous view on locations for location-based services. These services map raw location information from the positing drivers to both physical as well as global unique semantic locations. These services are described in [17].
- *Semantic geocast*: With this component, a location-based service can send and receive geocast messages.

The *server segment* contains the location servers that store the domain data. In principle, we could use one huge database and store hierarchies with the corresponding domains on a single server. One database for a huge number of potential clients, however, would be a bottleneck. In addition, information about local domains is usually available only locally and difficult to administrate in a central database. As a solution, we use a distributed system of location server each storing a number of domains. Each location server is responsible for a specific domain and all subdomains, for which no other location server exists. In our example, the location server for `hagen.de` covers `fley.hagen.de` and `downtown.hagen.de`, but not `university.hagen.de`, as this domain has its own location server.

The entire system is self-organizing. The location servers establish the relation and association links among each other automatically. Building these links is done by a set of lookup and discovery protocols not described in this paper (see [17] for details). In order to decouple the infrastructure from communication aspects, we use a communication middleware, especially designed for mobile scenarios [15]. When a mobile node moves to another location, it automatically looks up an appropriate location server (called the *local location server*, *LLS*). The LLS is the representative of the entire infrastructure for a mobile node. Any service usage is directed to the LLS. As mobile users are distributed among different location servers, this infrastructure is highly scalable. Especially, our system does not overload top-level servers.

4 Semantic Geocast using LSI

The logical structure of relations and associations forms an ideal platform for a geocast mechanism as domain information can be distributed among this logical network. A geocast request *r* from a mobile node contains a target domain *r.domain* and a message *r.message*. The goal is to transfer *r.message* to all mobile nodes residing at positions $p \in r.domain.c$.

In the following, we make a simplification: we represent every domain by its own location server. We assume that a communication between two domains always needs a network transaction. In reality, the performance of our system is far better, as communication often can be done inside a location server. Thus, our performance evaluations in a later section describe a worst-case scenario.

4.1 The Semantic Geocast Mechanism

The basic idea of our semantic geocast mechanism is as follows:

- *Registration*: Each mobile node registers itself at all location servers, which cover the occupied semantic locations. The location servers accept geocast requests and in turn deliver other geocast messages to the mobile nodes.
- *Address Propagation*: Each location server builds a list of network addresses of other location servers. The lists are periodically updated, thus, they notice, when servers start up or are shut down.
- *Message Passing*: When a location server receives a geocast request, it looks up an appropriate location server in its address list for delivery and redirects the request. Often, this server is not the final destination, thus additional transfers may be required.
- *Delivery*: Finally, a target location server receives the message and distributes it to the registered mobile nodes. As more than one location server may cover the target domain, additional transfers to other servers may be required.

Fig. 3 illustrates the basic mechanism. Note that in this figure, we equate domains to their location servers.

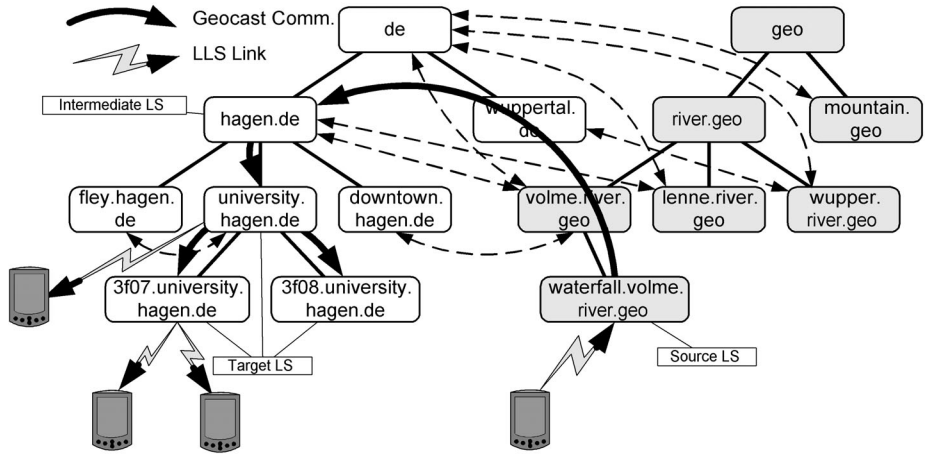

Fig. 3. The geocast mechanism

In this scenario, a mobile node residing at `waterfall.volme.river.geo` sends a geocast message to all mobile nodes at `university.hagen.de`. We distinguish the following servers, which process a geocast request:

- The *source LS* accepts the geocast request from a mobile node (`waterfall.volme.river.geo` in fig. 3).
- *Intermediate LSs* relay geocast requests to the target domains (`hagen.de` in fig. 3).
- *Target LSs* are responsible for the target domain and send geocast messages to the mobile nodes (`university.hagen.de`, `3.f07.university.hagen.de` and `3f08.university.hagen.de` in fig. 3).

Note that the communication between LSs can use "short cuts" and are not restricted to associations and relations. We assume that all servers are connected via a global network (usually the worldwide Internet). Once a server has another server in its address list, they can communicate directly.

The geocast mechanism mainly contains two parts: a proactive part to collect addresses and a reactive part to route geocast requests.

4.2 Address Propagation

Each server proactively collects addresses of other locations servers. This is permanently done in the background, thus new servers are registered after a delay. As the list of all location servers in the system can be very large (e.g., many thousands entries), we allow each server to build a list with a specific length limitation. This reduces the amount of memory required by an LS, but also reduces the traffic to maintain the address lists. Our mechanism ensures stability, even if a target LS is not listed by the source LS. Locations servers collect addresses according to two mechanisms: *slow propagation* and *fast propagation*.

The *slow propagation* is built according to the propagation mechanism integrated in the DSDV ad-hoc routing algorithm [13]. When a server starts up, it sends an

update message with its own address to its "neighbours", i.e. its master, all subdomains and all associated servers. The message contains a sequence number, which a server has to increase at every new start-up.

Whenever a server receives an address update, it first looks in its own table whether it already has received an update with this sequence number. If yes, the message is simply ignored; if not, it stores this new information and forwards the update to all neighbours apart from the originator. The sequence number avoids eternally circulating updates. Each server periodically (e.g. every day) increases its own address sequence number and distributes the address. Each address entry has a certain lifetime, specified by the originating server. Thus, disconnected servers are removed from the lists after the lifetime expired.

To reduce the overall traffic, each server collects update messages for a specific time (e.g. 10 minutes) and then exchanges them in a bundle. As a starting server does immediately flood updates through the network, denial of service attacks are more difficult. We call this mechanism the *slow propagation*, since it takes a considerable long time (e.g., some hours) for every location server to list an address of a new server.

To propagate new addresses much faster, we use an additional mechanism, the *fast propagation*. A new server first starts with the slow propagation. Its own address list initially is empty and thus it receives new addresses from its neighbours. Whenever it receives a root domain server, it uses the fast propagation mechanism: it *once* sends an address update to this root server. As a result, this root server distributes the new address in its own hierarchy, passing this information down the hierarchy tree. If neighbours of a new server already know the root domains of all hierarchies, addresses are distributed very fast among the entire infrastructure.

To investigate the effectiveness of our mechanisms, we ran a number of simulations. Since LSI is fully implemented and operable, we can use the real infrastructure for evaluation purposes. We developed an additional simulation tool to randomly generate a huge number of domains. The tool first creates a root domain for every hierarchy and then additional levels of domains by adding up to 10 subdomains for each domain. The process runs until we reach the required number of domains. Finally, the tool randomly adds associations between the hierarchies. We run a number of simulations with the same parameters to compensate outliers. We first use the random hierarchies to compare the slow and fast propagation (fig. 4). In the following, h denotes the number of hierarchies and n the total number of domains in all hierarchies.

As real network delays heavily depend on the actual network structure and load, we only measure the hops in our simulations. Fig. 4 shows the maximum number of hops to inform a server about a newly started server. We simulate scenarios with 2 and 16 hierarchies. If all nodes are distributed among a higher number of hierarchies, the propagation works more effectively, because associations connect domains more tightly. For any number of hierarchies, the fast propagation needs a significant lower number of hops to inform all domains, thus we always use fast propagation, whenever a new node collects a root server address.

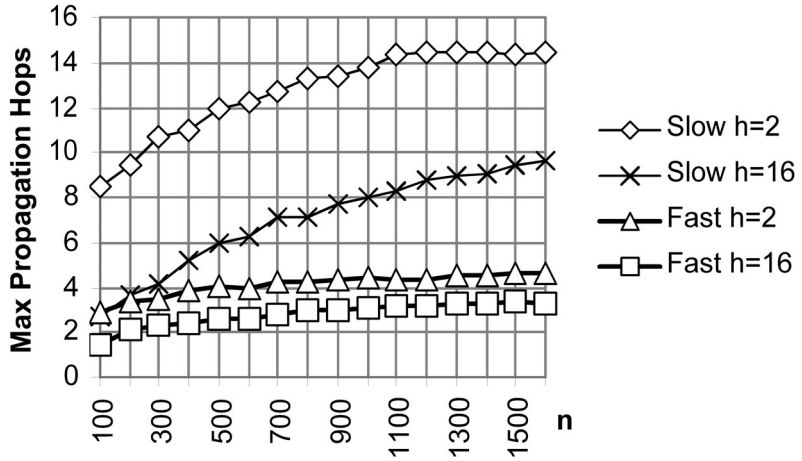

Fig. 4. Comparison of Slow and Fast Propagation

4.3 Message Passing

Once a source LS receives a geocast request from a mobile node, it looks up an appropriate location server. This location server can either be a target LS or an intermediate LS. The latter case occurs, if a target LS is not listed, either because of a limited list space or the propagation has not yet completed.

The following pseudo code outlines the thread integrated in every LS to handle geocast requests. Here, x denotes the local LS.

```
while (true) // The handle thread loops endlessly {
  wait for geocast request r;
  if r.domain≻x.domain { // I'm a target LLS ❶
    send r.message to all registered mobile nodes;
    send r to all subdomains;
  }
  else { // I'm an intermediate or a source LS ❷
    look up servers s in the local address list where
       s.domain≻r.domain – if more than one server is found,
       choose the lowest in the hierarchy;
    if such a server s is found
      send r to s;
    else { // Try routing via relations and associations ❸
      look up subdomain server y of x with y.domain≻r.domain;
      if such a server y is found  // there can only be one
        send r to y;
      else if x has associations into the target hierarchy {
        choose an appropriate associated server z
            (see selection above);
        send r to z;
      }
      else if x has a master m  // Try the master
```

```
          send r to m;
        else
          return an error to the originating node;
      }
  }
```

Usually a source LS handles a request (at ❷), which is directly passed to a target LS (at ❶). If a target is not listed, a request is send to an intermediate LS (at ❷), which may relay it in turn to another intermediate LS. The message passing mechanisms ensures that requests finally arrive at a target LS. A target LS delivers the messages to mobile nodes. In addition, it relays the request to all subdomains. Note that as subdomains are completely inside a master domain, every subdomain has to process the geocast request as well. Section ❸ contains a backup strategy, if address lists do not contain the required entries. In this case, a server asks its subdomains, its associated domains and its master to pass a request nearer to a target. If address lists contain a minimum of entries (see below), this block usually is not processed.

4.4 Dealing with Restrictions, Scalability

On one hand, our infrastructure should be scalable for a higher number (e.g., many thousands) of domains. On the other hand, each location server should be very lightweight, i.e. should not make high hardware demands. As a result, a location server could have a limited memory space for storing addresses in its list. Let l denote the maximum number of entries in the address list. To simplify the evaluation, we assume that each location server has the same space limitation. In reality, however, top-level servers may be prepared for larger lists. Our mechanism collects addresses using priorities: *1* (highest): root domains; *2*: all domains of the own hierarchy; *3*: all domains of other hierarchies. Domains with priorities 2 and 3 are further ordered by the domain level (higher levels first). The address list is filled according to these priorities: if the list is full and a new address is added, the entry with the lowest priority is dropped. For successfully passing geocast messages, at least a list of all root domains (except for the own) is necessary, thus $l \geq h\text{-}1$. Note that we do not store related or associated servers in the address list, as these links use a separate storage.

The second step ensures that in case of sufficient address space, at least all servers of a hierarchy know all servers of their own hierarchy. Thus, whenever a geocast request was directed to the target hierarchy, only one more hop is required to reach a target LS. The third step finally fills the list with domains of other hierarchies.

Fig. 5 shows the result of evaluating the message passing algorithm. Here, we count the maximum and average hops to reach the *first* target LS. The x-axis shows the limited list space in relation to the total number of domains. Not surprisingly, the average number of hops converges to 1 for higher l. Each curve has a certain break point (e.g. at 50% for h=2). At this point, address lists are capable to store all domains of the own hierarchy.

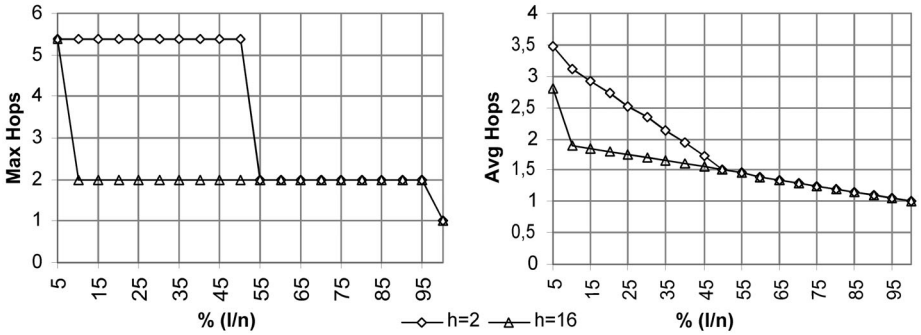

Fig. 5. Message passing with reduced address tables (n=1000)

As a result, the maximum hop count to reach a target is 2: one to reach the root server of the target hierarchy and one more to reach the target LS inside the hierarchy. Assuming n_h domains for each hierarchy, where $n_h \approx n/h$, we get a maximum of two hops for $l \geq h\text{-}1+n/h$. Having less entries in the address list, but not less than h-1, message passing still is successful, but the number of hops is considerable high.

4.5 Security Issues

Distributed systems, especially in mobile computing environments, are subject to security issues. Our security solutions are very complex, thus we can only present the ideas at this place. To protect a server or mobile node against malicious servers, a node can request an authentication certificate of the correspondence node. Authentication is proofed according to the challenge-response mechanism. A server can reject any request, if the authentication fails. This includes geocast requests as well as requests to register as a master, subdomain or associated server.

If mobile users do not want to receive any geocast message, they can register themselves in *stealth mode*. In this case, LSI only provides basic services. As the mobile user is not listed by an LLS, the system does not collect any position data. Note that protecting mobile users against malicious location servers that collect motion profiles generally is very difficult. The same unsolved problem occurs in cellphone networks. As our system is decentralized, such servers would have to cover a large area to capture motion profiles of mobile users.

A mobile user, who wants to receive geocast messages, is not willing to receive unwanted (i.e. spam) messages. As in traditional networks, an application listens for a set of pre-defined ports and ignores all messages arriving at other ports. In addition, our system offers mechanisms to discover the identity of a mobile user. As the mobile user cannot send geocast messages directly, but has to use an LLS, LSI can request a certificate of the sender for each geocast message. This information can be passed through to all receivers. We are aware that this mechanism cannot avoid spam completely, but even in traditional networks this problems is not solved.

4.6 Further Details

As mentioned above, LSI and its semantic geocast mechanism is fully implemented and tested. The following code shows how to send a geocast message with only a few lines of code:

```
LSI.startService();              // Start the runtime system
LSI.setStealthMode(false);       // I want to receive messages
byte[] msg="hello".getBytes();   // Create a message
LSI.sendGeocast(NATIVEGEOCAST,   // And send it to
     MSG_PORT,"hagen.de",msg);   // all nodes in Hagen
```

We distinguish two kinds of geocast requests: *native* requests use UDP for the last hop, i.e. from a target LS to mobile nodes. Using native requests, a receiver has to listen to a traditional UDP port to receive geocast messages. In contrast, *event-based* requests use internal protocols between the client and the LLS. Using the event-based mechanism, an application can either call a `receiveGeocast` method to wait for geocast messages or register a listener object that is called when a message arrived.

5 Conclusion and Future Work

In this paper we presented a decentralized, self-organizing approach to provide geocast services. We especially introduced the notion of *semantic geocast*, where target regions are defined by their meaning rather than by their physical area. We presented mechanisms that ensure scalability and stability, even if the servers have certain limitations concerning memory space.

LSI mainly addresses technical issues and provides a basic communication platform for location-based services. To use it in real environments, we additionally have to address organisational issues, e.g., we have to define useful hierarchies with meaningful domains. If LSI is a service inside a commercial infrastructure, e.g. a cellphone network, we need a system to charge users. Such organisational issues will be addressed in the future.

References

1. Abowd, G. D.; Atkeson, C. G.; Hong, J.; Long, S.; Kooper, R.; Pinkerton, M, 1997: Cyberguide: A mobile context-aware tour guide. ACM Wireless Networks, 3: 421-433
2. Cheverst, K.; Davies, N.; Mitchell, K.; Friday, A.; Efstratiou, C., 2000: Developing a Context-aware Electronic Tourist Guide, in Proc. of CHI'00, ACM Press
3. Dey, A., K.; Abowd, G., D., 2000: CybreMinder: A Context-aware System for Supporting Reminders, Second International Symposion on Handheld and Ubiquitous Computing 2000 (HUC2K), Bristol (UK), Sept. 25-27, 2000, LNCS 1927, Springer-Verlag, 187-199
4. Hohl, F; Kubach, U.; Leonhardi, A.; Schwehm, M.; Rothermel, K.: Nexus - an open global infrastructure for spatial-aware applications. In Proc. of the 5th Intern. Conference on Mobile Computing and Networking (MobiCom '99), Seattle, WA, USA, 1999. ACM Press
5. Imielinski, T.; Navas, J.: GPS-based Addressing and Routing, Request for Comments 2009, Nov. 1996, http://www.ietf.org/

6. Imielinski, T.; Navas, J.: GPS-based geographic addressing, routing, and resource discovery, Communications of the ACM, Vol. 42, No. 4, Apr. 1999, 86-92
7. José, R.; Davies, N.: Scalable and Flexible Location-Based Services for Ubiquitous Information Access, Proc. of the first Intern. Symposium on Handheld and Ubiquitous Computing HUC '99, Karlsruhe Germany, Springer-Verlag, 1999, 52-66
8. Kindberg, T.; Barton, J.; Morgan, J.; Becker G.; Caswell, D.; Debaty, P.; Gopal, G.; Frid, M.; Krishnan, V.; Morris, H.; Schettino, J.; Serra, B.; Spasojevic, M., 2000: People, Places, Things: Web Presence for the Real World, Proc. 3rd Annual Wireless and Mobile Computer Systems and Applications, Monterey CA, USA, Dec. 2000
9. Leonhardt, U.: Supporting Location-Awareness in Open Distributed Systems, PhD Thesis, University of London, 1998
10. Marmasse, N.; Schmandt, C., 2000: Location-aware Information Delivery with ComMotion, Second International Symposion on Handheld and Ubiquitous Computing 2000 (HUC2K), Bristol (UK), Sept. 25-27, 2000, LNCS 1927, Springer, 157-171
11. Navas, J.; Imielinski, T.: GeoCast – Geographic addressing and routing, Proc. of the 3rd ACM/IEEE inter. conf. on Mobile Computing and networking, Sept. 26-30, 1997, 66-76
12. Open GIS Consortium, OpenLS Home Page, www.openls.org
13. Perkins, C. E.; Bhagwat P.: Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers, Proc. of the SIGCOMM '94 Conference on Communications, Architectures Protocols and Applications, Aug. 1994, 234-244
14. Pradhan, S.: Semantic Locations, Personal Technologies, Vol. 4, No. 4, 2000, 213-216
15. Roth, J.: A Communication Middleware for Mobile and Ad-hoc Scenarios, Int. Conf. on Internet Computing (IC'02), June 24-27, 2002, Las Vegas, Vol. I, CSREA press, 77-84
16. Roth, J.: Context-aware Web Applications Using the PinPoint Infrastructure, IADIS Intern. Conference WWW/Internet 2002, Lisbon, Portugal, Nov. 13-15 2002, IADIS press, 3-10
17. Roth, J.: Flexible Positioning for Location-Based Services, IADIS International Conference e-Society 2003, Lisbon, Portugal, 3-6 June 2003, IADIS Press, 296-304
18. Schilit, B.; Adams, N.; Want, R., 1994: Context-Aware Computing Applications, Workshop on Mobile Computing Systems and Applications, Santa Cruz, CA, USA, 1994