IADIS International Journal on Computer Science and Information Systems
Vol. 1, No. 2, pp. 1-14
ISSN: 1646-3692

THE DISTRIBUTED LOCATION RESOLUTION PROBLEM AND ITS EFFICIENT SOLUTION

Jörg Roth *University of Applied Sciences Nuremberg, 90489 Nuremberg, Germany*

ABSTRACT

Location-based applications and services become increasingly important for mobile users. They take into account a mobile user's current location and provide a location-dependent output. To support developers of location-based services, the Nimbus framework hides specific details of positioning systems and provides uniform output containing physical as well as symbolic location information, which often are more suitable for applications and users. One basic problem is the *Distributed Location Resolution Problem*: given a location; which identifiable areas cover this location, if area information is distributed among different servers? This paper presents an algorithm that solves this problem and runs in the distributed, self-organizing Nimbus infrastructure.

KEYWORDS

Location-based service, symbolic location, self-organizing infrastructure

1. INTRODUCTION

The answer to the question "*Where am I?*" becomes increasingly important for mobile users. In the future, it will not only be sufficient to know the coordinates of the current location, but also to know information *about* the current location. Typical questions will be: "*Which bus takes me from here to the railway station?*" or "*Who of my friends are currently in this restaurant?*". Over the last years, many researchers developed so-called *location-aware applications* that respond to such questions. For such services, experts expect a huge market potential of several billion dollars in revenue worldwide in the next years [6, 16].

Location-based services often need *symbolic* location information, but most positioning systems only provide *physical* locations. In this paper we present the Nimbus platform, which provides a solution for the *distributed location resolution problem* (*DLRP*): given a location and a set of *distributed* computers, each storing a certain set of *identifiable areas*; which areas contain this location? Nimbus introduces a naming scheme and classifies these areas according to hierarchies. If an application gets well-defined area names for a certain location, it can easily lookup location-based information in e.g. databases or file systems. Providing these

location information, the framework enables applications in the area of mobile and pervasive computing which otherwise were not conceivable or at least very cost-intensive. The following examples motivate the benefits of this approach.

A universal location-based remote control is illustrated in fig. 1 (left). It offers different services depending on the user's current location: at home, it is a traditional remote control for home-entertainment equipment. At work, it is used as an electronic key to open office doors and on travel, it can be used to pay tickets for public transportation. Such a device is not primarily interested in the physical location but, in the type of the current position (e.g. home or office). Fig. 1 (right) presents a location-based bus planner, developed with the Nimbus framework. The user simply selects the destination and the application computes the appropriate timetable for the selected destination, while taking into account the current time and location. After boarding the bus, the planner supervises the current location and informs the user when to exit. Again, this application is not primarily interested in the physical coordinates of the user's current location, but wants to know at which bus station the user is waiting. As the bus station locations are stored decentrally (e.g. inside different bus station registers for the respective city), we have to look up these data in a distributed manner.

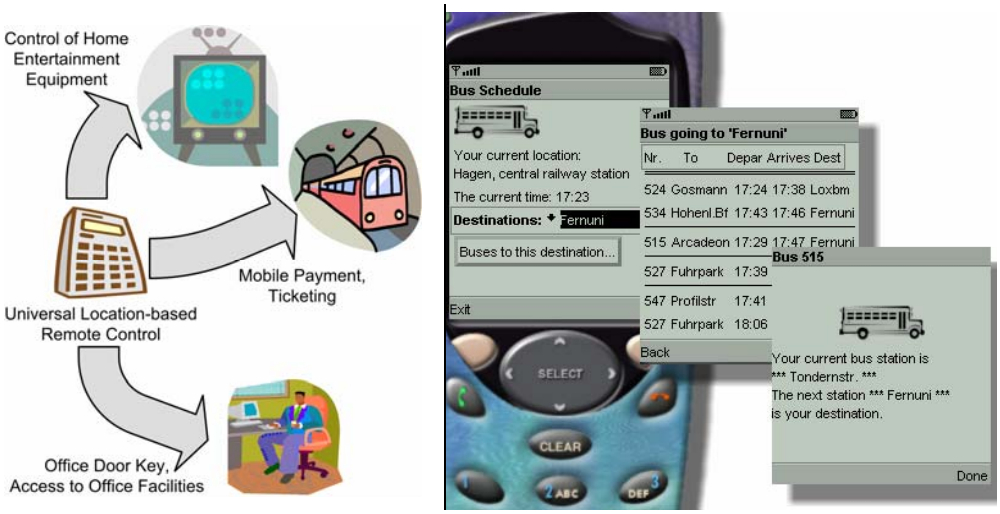

Figure 1. Location-based applications: the universal remote control (left), the Nimbus bus planner (right)

Areas such as bus stations and offices, but also streets, forests, city centres etc. are often called *symbolic* or *semantic locations* [8, 10, 14]. People expect a lot of semantics and intelligence behind these words, even though the term semantic location in its original meaning only describes an area that has a certain identity for a user or application. Thus, in Nimbus we use the term *identifiable area* instead [13]. Such an area can uniquely be identified by a *name*. The naming scheme provides a rough classification of the area, but the actual semantics behind it is only perceptible inside the respective application. In principle, one huge database on a single server could store all identifiable areas, but a single database would be a bottleneck. More reasonably, a mobile client has local access to the service. Moreover, information about local areas is usually available locally and is difficult to administrate in a central database. As a solution, Nimbus provides a distributed system of *Location Servers* that store the information

about identifiable areas in a decentralized manner. Location servers are connected to each other in a peer-to-peer style and provide a solution of the DLRP.

2. RELATED WORK

Several frameworks deal with location data and provide a platform for location-based application. E.g., the *Open Geospatial Consortium* [9] provides a high-level framework to build location-based services. The *LORE* platform [2] contains three key components: a location server providing a location sensing API, a moving object database and a spatial publish/subscription engine. All components are built according to a traditional client/server architecture. *Nexus* [5] introduces so-called *augmented areas* to formalize location information. Augmented areas represent spatially limited areas, which may contain real as well as virtual objects, where the latter can only be modified through the Nexus system. Inside the Nexus framework, different locations can be modeled in the so-called *augmented world model*: static objects like houses or streets, mobile objects such as users or cars and virtual objects such as virtual post-its or virtual kiosks. Between these locations, relations can be defined which express, e.g., inclusion of areas. The *RAUM* location model [1] was designed to address the needs of networked ubiquitous devices such as intelligent cups and memo clips. In its first version it could only store locations without any geographic information. In a second version, geometric information was added in order to express positions of objects.

A number of platforms focus on location sensing. The *Location Stack* [4] provides a layered approach to access multiple positioning systems. A similar approach follows the *POS* component inside the *PoLoS* project [11]. Specific positioning systems are modeled using wrappers. Even though platforms such as Location Stack and PoLoS simplify the process of location sensing, they do not consider the distributed nature of information *about* locations and focus on the location sensing and fusion function.

Geographic information systems (*GIS*) and spatial databases provide powerful mechanisms to store and retrieve location data [15]. They concentrate on accessing large amounts of spatial data. In our scenarios, however, we have to address issues such as connectivity across a network and mobility of clients, thus we have to use data distribution concepts, which are only rarely incorporated into existing GIS approaches.

Even though some of the platforms above provide services to discover identifiable areas, they do not solve the DLRP in its general meaning. E.g. in Nexus, a requesting client has to discover the appropriate server for the specific area via a *central* register. Requests concerning multiple servers have to be carried out one after the other. LORE is built according to a fully centralized client/server architecture and does not support any distributed storage of location information. In the following, we describe a fully distributed infrastructure that efficiently solves the DLRP without any central instance.

3. NIMBUS

The Nimbus framework supports developers of location-aware applications. The primary goal is to provide the most enriched location information including identifiable areas. Once the enriched location information is delivered to the application, the mobile client can use arbitrary

mechanisms to execute the location-based service. Established mechanisms such as databases, component services, web services or simple socket connections can be used to get further information about the location (fig. 2).

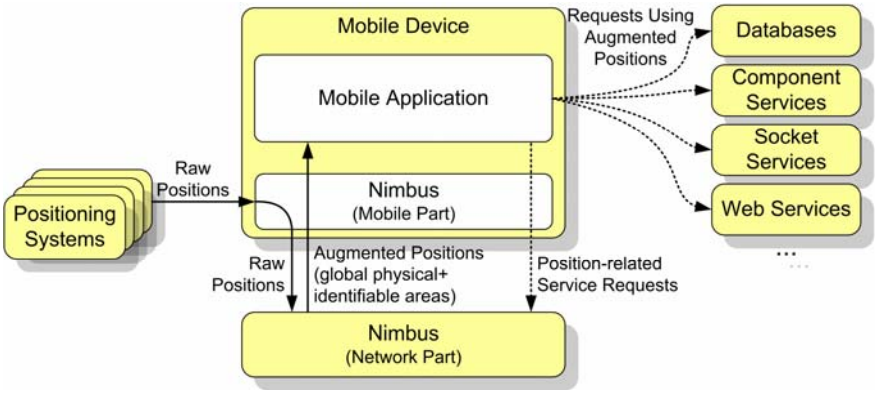

Figura 2. Data flow in the Nimbus framework

The Nimbus framework has a mobile part installed on the mobile system and a network part providing information about the location. The mobile part takes raw positions and requests the network part to provide an *augmentation*: it maps any local coordinates to WGS84 coordinates [3] and generates identifiable area information, which now can easily be processed by the application. Using a predefined name scheme, identifiable areas can be directly used for database queries or as parts of file names. Physical locations are still useful for all kinds of geometric queries, e.g. asking for distances between locations or asking for directions. Nimbus provides additional services related to location-based services (e.g. geocasting) not presented in this paper.

3.1 The Nimbus Location Model

We want to describe the concept of identifiable areas in Nimbus more precisely. Let P denote the set of all physical locations inside the observed area (e.g. all WGS 84 coordinates from a certain depth up to a certain altitude in the atmosphere). We call an area $S \subseteq P$ with a certain meaning an *identifiable area* of P. To identify areas not only by their geometry, they have a *name*. An area with its corresponding name is called a *domain*. For a domain d, *d.name* denotes the domain name, *d.c* the domain geometry. Further D denotes the set of all domains.

Names of identifiable areas could in principle be human-understandable, but there is no *single* human-understandable representation which is suitable for *all* conceivable applications. Thus, the area names are optimized for the internal Nimbus operations. Once the name space is clearly defined, transforming a name to its final representation often means simply looking up a string inside a table.

Looking at real-world scenarios space is often structured hierarchically, e.g. a room is inside a building, a building is in a city and a city is in a country. Thus, we divide the space into *hierarchies*. A hierarchy contains domains with a similar meaning, e.g. domains of cities or geographical domains. Each hierarchy has a *root domain* and a number of *subdomains*; each

of them can in turn be divided into subdomains. A top node of a subhierarchy is called a *master* of the corresponding subdomains. A link between a subdomain and its master is called a *relation*; master and subdomain are called *related*, denoted *master* ▷ *subdomain*. Names of subdomains are built similar to Internet host names, i.e. `<sub>.<master>`. Fig. 3 presents example hierarchies.

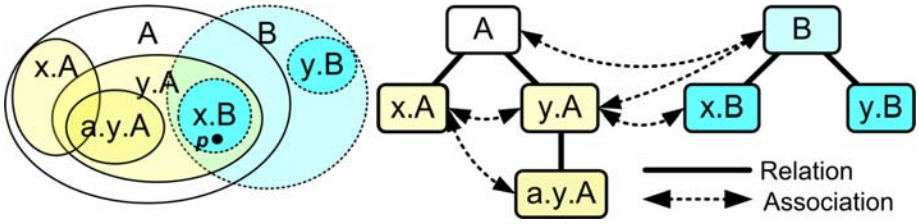

Figure 3. Hierarchies in the Nimbus framework

The Distributed Location Resolution Problem

As our initial goal, we want to solve the *distributed location resolution problem (DLRP)*:

> Given a location *p* and a set of distributed computers, each storing a certain set of identifiable areas $d_i.c$. Which areas (identified by their name $d_i.name$) contain *p*?

In fig. 3 point *p* resides in the domains `A`, `y.A`, `x.B` and `B`. Since a master always fully encloses a subdomain, the results `A` and `B` do not carry additional information. A sufficient answer to the question above is "`y.A` and `x.B`". Browsing through all hierarchies from the root down to the smallest domains containing *p* would cause a large number of requests and network traffic in distributed infrastructures. Therefore, a second relationship between domains is introduced, the *association*. Two domains d_1, d_2 are associated, denoted $d_1 \sim d_2$, if they are not related and their shared area is not fully covered by their respective subdomains. We introduce the abbreviation

$$\Delta(d) = \left(d.c \setminus \bigcup_{e \in D, d \triangleright e} e.c \right) \tag{1}$$

for the domain's area excluding the subdomains' area and get the definition

$$d_1 \sim d_2 : \Leftrightarrow \triangle(d_1) \cap \triangle(d_2) \neq \{\}. \tag{2}$$

In fig. 3 the area shared by `A` and `x.B` is fully covered by the domain `y.A`, thus `A` and `x.B` are not associated since this link would not provide additional information. Starting at `x.B`, only `y.A` has to be checked. With the help of associations, we now sketch a solution for the DLRP:

$DLRP(p) \rightarrow$ set of names
look up an arbitrary domain d_0 with $p \in \triangle(d_0)$
$names \leftarrow \{d_0.name\}$
for all $d \sim d_0$ do
if $p \in \triangle(d)$
$names \leftarrow names \cup \{d.name\}$

If an arbitrary domain d_0 is available that fulfils the first condition, the algorithm can efficiently loop through the associated domains. Note that the algorithm assumes that at every conceivable location, at least one domain d_0 is available that covers this location. This is usually true for real scenarios. We conducted a formal proof of correctness of this algorithm [12].

3.3 Looking up Location Servers

The question is how to find a domain d_0. To answer this question we now have to switch from the model view to the infrastructural view of Nimbus in which the domains are stored on location servers. Each location server can store a certain amount of domains, called a *cluster*. Each location server is responsible for a specific domain and all subdomains, for which no other location server is set up. Associations, relations and $\triangle$ areas can be defined for clusters and thus for servers in a canonical manner: two clusters are related, if there exist domains of each cluster that are related. An analogous definition can be made for associations. A $\triangle$ area is the cluster area without the subservers' cluster areas. The relation between clusters forms the so-called *cluster tree*. Fig. 4 shows the infrastructure based on the storage of clusters. This example uses more realistic hierarchies imported from land survey offices (see section 4).

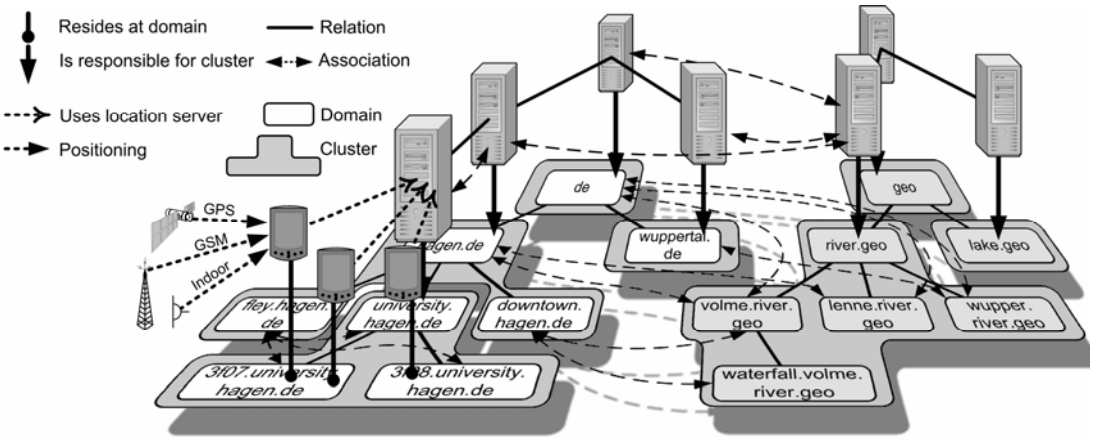

Figure 4. The distributed Nimbus infrastructure

One could argue that the communication to run the algorithm above is not acceptable. Fortunately, the infrastructure can proactively collect a lot of information to avoid heavy network traffic at resolution time: a location server that starts up, automatically discovers the

master server and all associated servers and performs a registration in order to exchange the △ areas. As a result, all information required by the algorithm above is stored locally on each server, also by the server that contains d_0. A client has only to discover that server, which then can execute the algorithm's main loop. In addition, a client can cache previous results to improve the execution time even more.

As mobile users are distributed among different location servers, this infrastructure is highly scalable. The approach does not overload top-level location servers. One can add a server and restructure the respective hierarchy, thus it is possible to add server power to the infrastructure in a simple way.

When a mobile node moves to a specific location, it automatically looks up an appropriate location server for the new location, called the *Local Location Server* (*LLS*). The LLS is the representative of the infrastructure for a mobile node. In particular, the LLS stores the domain d_0 of the DLRP algorithm. Usually, the LLS can fully answer the queries without any further network transactions. Inside the covered area, a user can move around without looking up a new location server. Only when a mobile user leaves a certain area, a new LLS has to be looked up. The lookup uses the following mechanisms:

- a mobile client receives information about the LLS from the positioning system,
- a mobile client performs a search using broadcasting lookups (using e.g., MBone or UDP broadcast), or
- a mobile client asks another location server (e.g. an old LLS) which replies a *redirect* message for a more suitable server.

The first mechanism is most effective, but requires adapting the positioning systems. Nimbus usually uses a combination of the latter two mechanisms. The client performs a lookup procedure as illustrated in fig. 5.

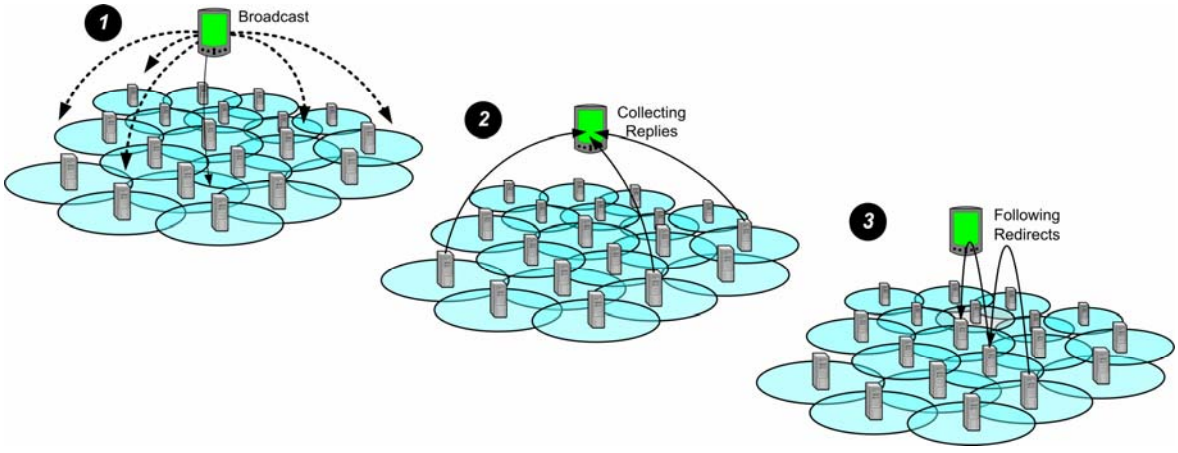

Figure 5. The lookup mechanism

The corresponding algorithm can be sketches as follows:

```
LLS_lookup → LLS
    askedservers ← {}; redirect ← set of former LLSs
    if redirect = {}
        send a broadcast message                    // (1) in fig 5.
        collect broadcast replies → redirect        // (2) in fig 5.
    while not LLS found   // i.e. no success message received
        return error, if no server in redirect\askedservers is available
        send a redirect request to the geometrically nearest server of
redirect\askedservers
        add this server to askedservers, remove from redirect
        add redirect reply to redirect              // (3) in fig 5.
```

This algorithm indirectly performs a backtracking, if a sequence of redirects gets caught in a dead end. In contrast to the resolution algorithm in section 3.2 which mainly uses the logical links between domains, the lookup mechanism is mainly based on their geometry. Logical links are only used to generate answers to clients: a server which receives a broadcast or a redirect request answers according to the following rules:

- If it is the LLS or knows the LLS as its master, subserver or associated server, it replies a *success* message.
- If one of its master, subserver or associated server is geometrically nearer to the client, it replies the nearest one in a *redirect* message.
- Otherwise no reply is performed.

It is important to note that a broadcast only is required during the first startup. Once connected, a client can lookup an LLS with the help of a redirect request to any other server. For this, clients cache addresses of former LLSs.

3.4 Avoiding Reply Storms

A lookup via broadcast may reach many servers. If every server answers, the resulting so-called *reply storm* could overload a mobile client or its wireless connection. To reduce the overall network traffic, a server replies a redirect with a certain probability $q_{redirect}$. For this purpose, each server that receives a lookup, generates a random number $rand \in [0, 1)$ and only if $rand < q_{redirect}$ a redirect message is replied.

We can use different functions that model $q_{redirect}$. To be useful, $q_{redirect}$ should fulfill certain properties. First, small distances should produce higher values than big ones in order to get more local answers. Second, the probability should be zero beyond a certain distance; otherwise even far away servers would sometimes be forced to reply. Third, the overall behavior should easily be controlled with the help of some variables. We examined two functions as sketched in fig. 6.

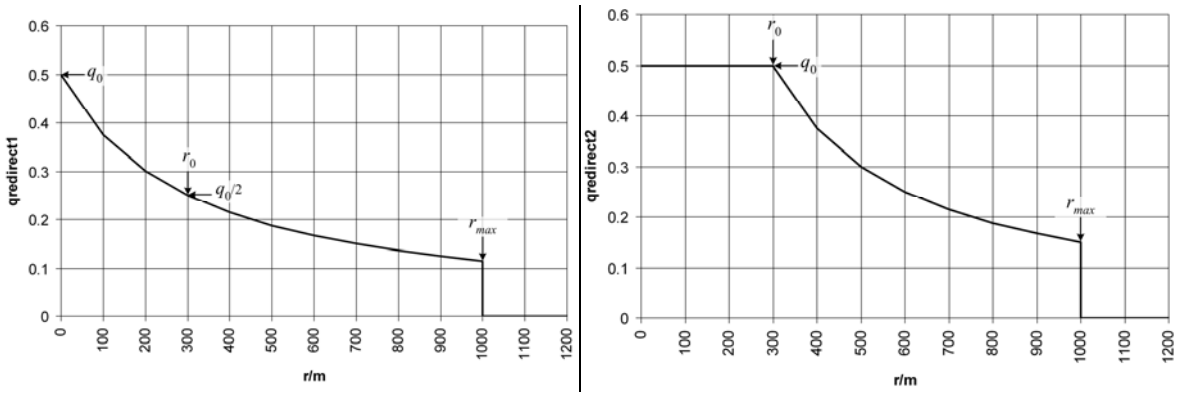

Figure 6. Possible functions for $q_{redirect}$

The first function for $q_{redirect}$ (fig. 6 left) is computed according to the formula

$$q_{redirect}(r) = \begin{cases} \dfrac{q_0}{\dfrac{r}{r_0} + 1} & \text{if } r \le r_{max} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

where r is the distance of the domain's border to the given location; $q_0 \in (0, 1]$ is the maximum value of $q_{redirect}$; $r_0 \ge 0$ is the distance that leads to $q_{redirect} = q_0/2$ and $r_{max} \ge r_0$ is the maximum distance to get $q_{redirect} > 0$.

Even though this function is simple to compute, the term +1 in the denominator leads to an undesirable behavior concerning the distribution of replies: as we do not have a uniform distribution among the distances, the number of replies is difficult to control. As a result we decided to use a second function (fig. 6 right), according to the formula

$$q_{redirect}(r) = \begin{cases} q_0 & \text{if } r < r_0 \\ q_0 r_0 / r & \text{if } r \ge r_0 \text{ and } r \le r_{max} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

where r again is the distance of the domain's border to the given location; $q_0 \in (0, 1]$ is the maximum value of $q_{redirect}$; $r_0 \ge 0$ is the maximum distance that leads to $q_{redirect} = q_0$ and $r_{max} \ge r_0$ is the maximum distance to get $q_{redirect} > 0$. For further considerations, we assume that r_{max} is significantly greater than r_0. A reasonable assignment is, e.g., $q_0 = 0.8$, $r_0 = 100$ m and $r_{max} = 1.5$ km. In the following, we use the formula (4) for $q_{redirect}$.

The function $q_{redirect}$ affects two quantities: the average number of broadcast replies and the average distance of the nearest replier. The client is interested in getting a sufficient number of replies without being overloaded. In addition, the minimum distance of all replies should be as near as possible. We want to explore some characteristics of $q_{redirect}$ which help to control its behavior.

The average number of areas completely inside or cut by a circle with radius r can be approximated by

$$n_{approx}(r) = \pi \frac{r^2}{\Delta} \cdot n_{ovr} + 2\pi \frac{r}{\sqrt{\Delta}} \cdot n_{ovr} \tag{5}$$

where $\triangle$ is the average size of all $\triangle$ areas and n_{ovr} the average number of overlapping $\triangle$ areas at a point in space. Then the expected number of broadcast replies according to our mechanism is

$$n_{reply} = q_{br} \int_0^{r_{max}} q_{redirect}(r) \left(\frac{2\pi \cdot n_{ovr}}{\Delta} r + \frac{2\pi \cdot n_{ovr}}{\sqrt{\Delta}} \right) dr = q_{br}\, 2\pi q_0 n_{ovr} r_0 \left(\frac{1}{\Delta} \left(r_{max} - \frac{r_0}{2} \right) + \frac{1}{\sqrt{\Delta}} \left(1 + \ln(\frac{r_{max}}{r_0}) \right) \right) \tag{6}$$

where q_{br} is the probability to reach a server with a broadcast message. Note that we assume a uniform distribution of hosts receiving a broadcast. In reality, it depends on network-related properties not modeled by this analytical approach. We further can approximate the average distance of repliers by

$$r_{replyavg} = \frac{1}{n_{approx}(r)} \int_0^{r_{max}} q_{redirect}(r) \left(\frac{2\pi \cdot n_{ovr}}{\Delta} r^2 + \frac{2\pi \cdot n_{ovr}}{\sqrt{\Delta}} r \right) dr = \frac{\left(\frac{1}{2} {r_{max}}^2 - \frac{1}{6} {r_0}^2 \right) + \sqrt{\Delta} \left(r_{max} - \frac{1}{2} r_0 \right)}{\left(r_{max} - \frac{1}{2} r_0 \right) + \sqrt{\Delta} \left(1 + \ln(\frac{r_{max}}{r_0}) \right)} \tag{7}$$

which does not depend on q_{br}. As the client uses the nearest answer for redirection, it is interested in the expected *nearest* distance $r_{replymin}$ of all n_{reply} replies. For arbitrary distributions, it is difficult or impossible to get a closed formula for this distance. In our case, however, we approximately have a uniform distribution of distances, which is one great benefit of our specific $q_{redirect}$, not proved in this paper. If further r_{max} is significantly greater than r_0, we thus can approximate $r_{replymin}$ by

$$r_{replymin} \approx \frac{r_{max}}{n_{reply} + 1} = \frac{\Delta \cdot r_{max}}{q_{br}\, 2\pi q_0 n_{ovr} r_0 \cdot ((r_{max} - r_0/2) + \sqrt{\Delta}(1 + \ln(r_{max}/r_0))) + 1/\Delta} \approx \frac{\Delta}{q_{br}\, 2\pi q_0 n_{ovr} r_0} \tag{8}$$

This approximation is surprisingly useful in reality as shown in our evaluations. At this point we can summarize two benefits of $q_{redirect}$. First, the client can easily control the average number of replies by r_{max}, which can be increased during runtime, if too few replies arrive on average. For greater r_{max}, we nearly get a linear dependency of r_{max} and n_{reply}. Second, the expected distance of the nearest reply remains constant (and low) for increasing r_{max}. As the number of redirect requests mainly depends on $r_{replymin}$, we thus get a low traffic for redirects.

3.5 Computing a Configuration

A further goal is to compute a concrete configuration for given target values of n_{reply} and $r_{replymin}$. As we have to determine three variables q_0, r_0 and r_{max}, but only two given values, we have to put two of the three variables into relation. A reasonable assumption is that $r_0 = r_{max}/15$, where the factor 15 is based on the evaluations shown below.

As a consequence

$$n_{reply} = q_{br} 2\pi q_0 n_{ovr} r_0 \cdot \left(\frac{14.5 r_0}{\Delta} + \frac{3.71}{\sqrt{\Delta}} \right) \tag{9}$$

and

$$n_{reply} \cdot r_{replymin} \approx q_{br} 2\pi q_0 n_{ovr} r_0 \cdot \left(\frac{14.5 r_0}{\Delta} + \frac{3.71}{\sqrt{\Delta}} \right) \cdot \frac{\Delta}{q_{br} 2\pi q_0 n_{ovr} r_0} = 14.5 r_0 + 3.71\sqrt{\Delta} \tag{10}$$

We thus get a configuration r_{0conf}, $r_{maxconf}$ and q_{0conf} that leads to target values of n_{reply} and $r_{replymin}$ as follows:

$$\begin{aligned} r_{0conf} &= (n_{reply} \cdot r_{replymin} - 3.71\sqrt{\Delta})/14.5 \\ r_{maxconf} &= 15 \cdot r_{0conf} \\ q_{0conf} &= \frac{\Delta}{r_{replymin} q_{br} 2\pi n_{ovr} r_{0conf}} \end{aligned} \tag{10}$$

Not all values of n_{reply} and $r_{replymin}$ lead to reasonable configurations, as at least two restrictions have to be considered: r_{0conf} has to be greater than zero and q_{0conf} must not be greater than one. This leads to two inequations:

$$\begin{aligned} n_{reply} \cdot r_{replymin} &> 3.71\sqrt{\Delta} \\ r_{replymin} q_{br} 2\pi n_{ovr} r_{0conf} &\geq \Delta \end{aligned} \tag{11}$$

Thus, a configuration can not fulfill any unrealistic setting such as $n_{reply} = 1$, $r_{replymin} = 0$ m as such values would produce invalid values of q_{0conf} and r_{0conf}.

As an example (based on the settings presented in the next chapter), we want to compute a configuration for $n_{reply} = 10$ and $r_{replymin} = 250$ m. These values fulfill the inequations (11), thus lead to a reasonable configuration. With the help of formulas (10) we get $r_{0conf} = 144.6$ m, $r_{maxconf} = 2169.0$ m and $q_{0conf} = 0.854$.

4. EVALUATIONS

The efficiency of the Nimbus mechanisms is evaluated with the help of several simulations which use real imported data from the German land survey office. In Germany, the ATKIS database provides geospatial data for governmental purposes [7]. For the Nimbus project, data covering an area of 84 km^2 in the area of Hagen was purchased. It contains approx. 80000 raw database records which were converted to approx. 8000 Nimbus domains, divided into three hierarchies: a hierarchy containing the city, parts of the city properties, sites and buildings, a hierarchy containing those domains with a geographical meaning, currently rivers and lakes, and a hierarchy containing roads and railways.

The location server application consists of approx. 4000 lines of Java code (basic libraries excluded). On a 2.8 GHz (Windows XP) computer, the location server storing approx. 1000 domains generates a response to a single resolution request in 9.7 ms on average (only counting the CPU time, not considering the network link). The first measurements thus consider several clients simultaneously requesting resolutions over the network. All caches were switched off. Once receiving a resolution result, the next request is sent without any delay. This causes a high load to the servers.

The servers were connected via a 100 MBit/s Ethernet switched network. The delay for wireless phone connections would be higher, but with higher delays the measured effects are diluted and the measurement result would be less expressive. In addition, it is more difficult to impose load on location servers using slow connections. Location servers are Windows XP computers with a CPU speed between 1.8 and 2.8 GHz.

The experiments are conducted for 1, 2 and 3 servers (fig. 7, top left). Using more than one location server, the load of different users is distributed. As expected, the approach is scalable: the more location servers are used, the more flat is the curve. Whenever in reality a location server is overloaded, new subservers can easily be introduced. It is important to note that in reality a much higher number of users per location server is needed to cause a specific load as clients use a cache and clients usually do not permanently perform resolution requests. As an example, we assume that an application needs one location request per second and, as a result of using a cache, only one of five resolution requests are carried out over the network. Instead of 100 users, we then get a total of 1000 users that impose the same load to a single location server.

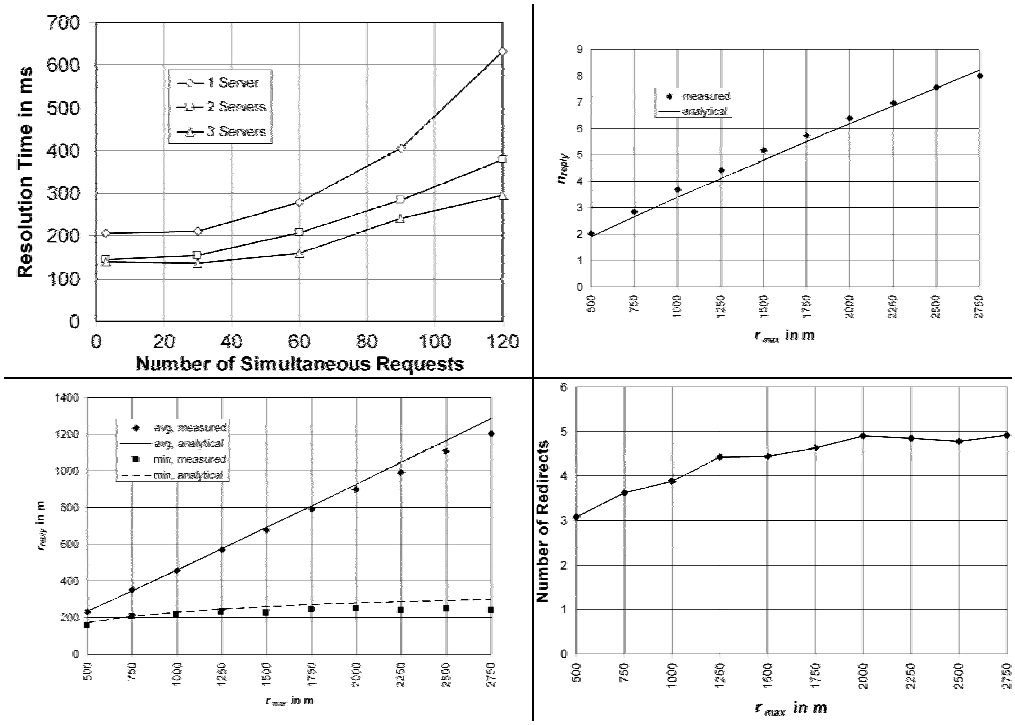

Figure 7. Evaluations of Nimbus characteristics

The next measurements present the effectiveness of the lookup mechanisms. As a worst case scenario we assume a separate server per domain. The imported data is characterized by $\triangle$ = 11817 m^2 and n_{ovr} = 1.218. We further assume that only 5% of servers nearer than r_{max} answer the broadcast, i.e. q_{br} = 0.05. The function $q_{redirect}$ is configured by r_0 = 100 m and q_0 = 0.8; r_{max} is variable. Fig. 7 (bottom left) presents the distance of replying servers. From the network view, two further numbers are important: replies per broadcast (fig. 7, top right) and the redirects to find the LLS (fig. 7, bottom right). The charts present numbers rather than network times as they are more meaningful.

As a first observation, the analysis presented in section 3.4 is very accurate even though the real data has large variations in, e.g., the size of $\triangle$ areas. Second, the lookup algorithm effectively avoids reply storms and on the other hand allows a client to control the number of broadcast replies in a fine-grained manner with the help of r_{max}. Third, even though the number of replies increases, the number of redirects to reach the LLS remains constant and low – for our measurements, less than five redirects were required on average.

5. CONCLUSION AND FUTURE WORK

The main goal of Nimbus is to offer enriched location information that is easy to process. Nimbus provides globally unique physical locations, independently of the underlying positioning systems and identifiable areas according to a predefined name space. Nimbus reached this goal with the following instruments: a location model which supports efficient

access to decentralized stored area data and a distributed, self-organizing infrastructure, easily to be administrated, designed according to scalability issues. Performance evaluations show the efficiency and scalability of the approach.

Even though Nimbus reached a high level of completeness, there are some open issues. In the current implementation, Nimbus stores domains with a public character. Every user can access all domains as, e.g., rivers or cities have a certain meaning for the public. Some domains however should not be open for everyone, e.g. barracks in a military area. We are currently working on appropriate access control mechanisms.

A second issue results in mobile domains: trains or ships, e.g., permanently change their location. In principle, Nimbus supports such domains, but such domains cause a high amount of updates messages. We currently work on a mechanism which avoids huge traffic and at the same time ensures consistency.

REFERENCES

1. Beigl, M.; Zimmer, T.; Decker, C.: A Location Model for Communicating and Processing of Context, *Personal and Ubiquitous Computing*, Vol. 6, No. 5-6, Dec. 2002, Springer-Verlag, 341-357
2. Chen, Y.; Chen, X. Y.; Rao, F. Y.; Yu, X. L.; Li, Y.; Liu, D.: LORE: An infrastructure to support location-aware services, *IBM Journal of Research & Development*, Vol. 48, No 5/6, Sept. 2004, 601-615
3. EUROCONTROL European Organization for the Safety of Air Navigation: WGS 84 - Implementation Manual, Brussels, Belgium, Febr. 1998
4. Hightower, J.; Brumitt, B.; Borriello, G.: The location stack: A layered model for location in ubiquitous computing, *Proc. of the 4th IEEE Workshop on mobile Computing Systems & Applications (WMCSA 2002)*, Callicoon, NY, USA, June 2002. IEEE Computer Society Press, 22-28
5. Hohl, F; Kubach, U.; Leonhardi, A.; Schwehm, M.; Rothermel, K.: Nexus - an open global infrastructure for spatial-aware applications, *Proc. of the 5th Intern. Conference on Mobile Computing and Networking (MobiCom '99)*, Seattle, WA, USA, 1999. ACM Press
6. Kelsey Group: Advertising and E-commerce, http://www.kelseygroup.com/, 2001
7. Land Survey Office North Rhine-Westphalia, http://www.lverma.nrw.de
8. Leonhardt, U.: Supporting Location-Awareness in Open Distributed Systems, PhD Thesis, Univ. of London, 1998
9. Open Geospatial Consortium, OGC Home Page, http://www.opengeospatial.org/
10. Pradhan, S.: Semantic Location, Personal Technologies, Vol. 4, No. 4, 2000, 213-216
11. Prigouris, N.; Papazafeiropoulos G.; Marias, I.; Hadjiefthymiades S.; Merakos, L.: Building a Generic Broker for Location Retrieval, *Proc. of the IST Mobile & Wireless Communications Summit*, Portugal, June 2003
12. Roth, J.: A Decentralized Location Service Providing Semantic Locations, *Informatics Report 323*, University of Hagen, Jan. 2005
13. Roth, J.: Detecting Identifiable Areas in Mobile Environments, Proc. of the 21st Annual ACM Symposium on Applied Computing, April 23-27, 2006, Dijon, (France), ACM Press, 986-991
14. Schilit, B.; Adams, N.; Want, R.: Context-Aware Computing Applications, *Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, USA, 1994
15. Tomlin, C., D.: Geographic Information Systems and Cartrographic Modeling, Prentice Hall, 1990
16. UMTS Forum: Enabling UMTS/Third Generation Services and Applications, Report 11, Oct. 2000, http://www.umts-forum.org