Master Thesis

Improving spatial indexing and searching for location-based DNS queries

Daniel Moscoviter

Supervisors

Dr. Ir. Geert Heijenk
Prof. Dr. Ir. Aiko Pras
Mozhdeh Gholibeigi, M.Sc.
Bernd Meijerink, M.Sc.

Ruben Kooijman, M.Sc.
Paul Krijger, M.Sc.

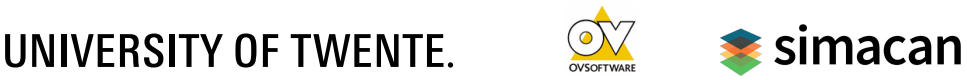
Faculty of Electrical Engineering,
Mathematics and Computer Science

Design and Analysis of
Communication Systems

University of Twente
Enschede, The Netherlands

OVSoftware B.V.
Oldenzaal, The Netherlands

Simacan B.V.
Amersfoort, The Netherlands



17th February, 2016

Contents

Abstract

In the domain of vehicular networking, it is desirable to be able to address vehicles based on geographical position rather than network address. The integration of geocasting (the dissemination of messages to all nodes within a specific geographical region) into the existing addressing scheme of the Internet has been a challenge. One solution to Internet-based geographical addressing is eDNS, an extension to the DNS protocol. It adds support for the querying of geographical locations as a supplement to logical domain names. In this thesis, eDNS is extended with support for querying various geographical shapes as well as nearest neighbor resolution, and a prototype server implementation is developed based on PowerDNS that uses bounding box propagation between servers for delegation.

Preface

This thesis is written for my Telematics master project at the University of Twente. It is carried out at OVSoftware B.V. in Oldenzaal, with strong cooperation of Simacan B.V. in Amersfoort. This document was written under supervision of Dr. Ir. Geert Heijenk, Prof. Dr. Ir. Aiko Pras, Mozhdeh Gholibeigi M.Sc. and Bernd Meijerink, M.Sc. from the Design and Analysis of Communication Systems (DACS) chair at the University of Twente, as well as Ruben Kooijman, M.Sc. and Paul Krijger, M.Sc. from Simacan.

Chapter 1

Introduction

The concept of Intelligent Transportation Systems (ITS) is an emerging area of research [26]. The goal of such systems is to use vehicular communication to develop novel applications for increasing safety, traffic management, Internet access, or other valuable services. Aside from the opportunities for delivering many novel applications, a significant amount of research has focused on ITS because of the technological difficulties that are involved. Mainly, vehicular networks have to deal with highly dynamic network topologies of vehicles, their high speed, limited communication ranges, and real-time constraints of potential applications. We can differentiate between two different types of communication in vehicular networks: the first is Vehicle-to-Vehicle (V2V) communication, and the second is Vehicle-to-Infrastructure (V2I). In the former type, data packets are exchanged between vehicles using vehicular communication technologies without involvement of an infrastructure. The latter type extends the vehicular ad hoc networks (VANETs) with a fixed infrastructure.

Vehicles in ITS will typically be equipped with localization technologies, such as Global Positioning System (GPS) receivers. This allows vehicles to be addressed based on their geographical positions rather than the network address (i.e. targeting a certain area, not a certain vehicle), utilizing domain-specific forwarding strategies [1]. Messages can be sent to any single node within a target region (geoanycast), or to all nodes within a target region (geocast). Geocasting in particular enables a large number of new applications. Warning about dangerous road conditions, assisting in speed management, and delivery of infotainment are examples of use cases that geocasting can facilitate.

Geocasting requires a routing protocol that delivers messages to the intended targets. It is a challenge to integrate geographically-scoped broadcasting into the existing addressing scheme of the Internet, as Internet Protocol (IP) networking based does not support geographical routing. This complicates VANET communication. Solutions to this problem have been

proposed, but as far as we are aware, all have shortcomings that are difficult to overcome. For instance, GPS-based addressing and routing [21, 22] requires a specialized infrastructure; use of a geographical IPv6 prefix format [15] relies on a standardized allocation of IPv6 addresses; and GeoNet [8, 27] has scalability limitations [11] on an Internet-wide scale.

One solution that was proposed because of these shortcomings is the Extended DNS (eDNS) [10, 11]. It is based on the Internet Domain Name System (DNS) protocol, and extends it to support geographical addressing. While DNS already supports a way to store locations, the novelty of eDNS is that the locations can be used as a primary key to return IP addresses that are associated with geographical regions. The appeal of this method is that it does not require specialized hardware or software, nor does it require protocol modifications. Only modification of existing DNS implementations is required, as described later in this document. Support for more efficient indexing, as well as delegation was added at a later stage [48]. However, opportunities for improvement remain. For example, it is currently only possible to request entities that are within a circular geographic region. Other shapes, such as polygons, could be used to more accurately retrieve information about entities on elongated target areas, such as highways. We can also consider use cases where one would be interested in entities close to a certain point, rather than entities within a region. More possible improvements can be identified, and will be discussed further in this report.

1.1 Research questions

As mentioned, the concept of integrating geocasting into the existing infrastructure of the Internet is an important development in ITS research. Because of the aforementioned shortcomings in several proposed solutions, this thesis focuses on the eDNS protocol as a solution for the problem of addressing all entities within a geographical area. This allows us to formulate our main research question.

- How can the functionality of eDNS be improved for use in vehicular networks?

To answer this research question, sub research questions were composed to be able to perform our research in a structured way. In addition, the sub research questions address the potential use of eDNS in products by Simacan.

- How can geographical areas with arbitrary shapes be addressed in eDNS?
- Can eDNS return nearby results when no results are found in a queried area?

- How can the implementation of eDNS in existing DNS name servers be simplified?
- Can eDNS be used for dynamic objects, such as moving vehicles?
- How can we evaluate the performance of eDNS operations?
- What is the performance of eDNS operations?
- What are potential use cases for eDNS to be used in Simacan products or services?
- Is the performance of eDNS adequate for a Simacan use case?
- What changes can be made to eDNS to facilitate widespread adoption?

1.2 Approach and Contributions

In this thesis, we have improved and evaluated the eDNS protocol in several ways. This has resulted in the following contributions.

- We have designed and implemented a method for querying lines and polygons, in addition to circles, as a way to address arbitrary areas.
- We have designed and implemented nearest neighbor resolution by introducing a new DNS resource record containing a location's distance to a queried area.
- We have abstracted the implementation of eDNS by using PostGIS for geographical calculations to simplify its integration in other DNS name server software.
- We have designed and implemented the propagation of bounding boxes to parent DNS name servers upon receiving location updates via the Dynamic DNS (DynDNS) protocol.
- We have evaluated the performance of our eDNS implementation in terms of throughput and latency for various input parameters.
- We have applied eDNS to a potential use case for Simacan and evaluated its suitability in terms of performance.
- We have evaluated changes required for eDNS to gain large-scale adoption.

1.3 Outline

This report is structured as follows. In Chapter 2, a background study is performed on the concepts and technologies that are can be used to answer our research questions. This includes VANETs, the DNS protocol, as well as the eDNS extension protocol.

Chapter 3 discusses the shortcomings that exist in the mentioned eDNS protocol, and how we plan to solve these problems. In addition, possible use cases for eDNS in Simacan will be identified. The design of our prototype eDNS system is discussed in Chapter 4. Several technological challenges are mentioned, as well as our solution to solve these challenges.

In Chapter 5, we evaluate the performance of our developed prototype with various combinations of setups and input parameters. Finally, we use our research in earlier chapters, and the results from our evaluation, to answer our research questions in Chapter 6.

Chapter 2

Background

To answer our research questions, it is necessary to discuss the context of the research first. We will look at VANETs and DNS, before exploring the current state of eDNS.

2.1 Vehicular ad hoc networks

VANETs are mobile networks that consist of vehicles. These vehicles are supposed to have On-Board Units (OBUs) that allow them to communicate with other vehicles in the network using wireless communication technologies. Messages can be sent between OBUs that are not directly in range of each other over a multi-hop chain. Among other types of wireless radio technologies, IEEE 802.11p [9] and Dedicated Short Range Communications (DSRC) [23] have been created specifically for the purpose of vehicular networking. The former has emerged as a defacto standard in the ITS context. Cellular, or other network connectivity types may also be supported. Vehicular networks have to deal with highly dynamic network topologies of vehicles, their high speed, limited communication ranges, real-time constraints of potential applications, and various other challenges [26].

We can differentiate between two types of communication in vehicular networks: V2V communication, and V2I communication. The latter requires the integration of stationary infrastructure into VANETs to create hybrid VANETs. ITS are systems that use vehicular communication to provide increased safety, traffic management, or other valuable services. Vehicles in VANETs will typically be equipped with localization technology. This allows for applications that do not send messages to specific nodes in a network, but rather to nodes based on geographical location. Dissemination of messages to all nodes within a geographical area, known as geocasting, allows for novel applications to be created. These will be categorized in Section 2.1.1. However, as we will discuss in Section 2.1.4, introducing geographical addressing in an Internet-based network is an open issue.

2.1.1 Applications

Numerous novel applications can be conceived that utilize ITS to achieve a specific goal. Schoch et al. [40] give an overview of application categories for vehicular ad hoc networks, compiled from multiple sources. We will briefly discuss them here.

Active safety

Active safety applications are used to increase the safety of driving using vehicular communication. This category is seen as the most important use case. The applications can either warn or instruct the driver of a vehicle, or instruct the vehicle directly to avoid or respond to accidents. Further subcategorization is possible based on the danger level. This can range from warning about dangerous road features, such as an upcoming hairpin turn (low danger), to preventing collisions, preparing a vehicle for an impending crash, and warning approaching vehicles when an incident has already occurred (high danger).

Public service

A vehicular network may aid public service work, such as emergency response. Notably, an emergency vehicle could announce its presence to vehicles on the emergency vehicle's planned route to give the other vehicles enough time to give way. Similarly, traffic lights could be preempted to allow an emergency vehicle to reach its destination faster.

Improved driving

Driving can be improved in terms of speed and simplicity. On a small scale, improved driving applications could assist a driver in merging into flowing traffic. On a large scale, drivers could be advised to change their speed to prevent shockwave traffic jams. A vehicle could also receive traffic light information, allowing it to calculate an optimal speed to reach the traffic light when the signal is green. Another possible application is route guidance, where a vehicle may be informed of delays or more optimal routes.

Mobile business and entertainment

Infotainment services can be provided to drivers. Examples of this include Internet access, advertising, parking management, and toll road payment.

2.1.2 Networking categories

A VANET may be infrastructure-less or infrastructure-based. Infrastructure-less networks allow for V2V communication by making use of ad hoc con-

nections between OBUs. A network based on a fixed infrastructure typically makes use of Roadside Units (RSUs), in addition to the OBUs. RSUs are usually connected to Back Offices (BOs) on the Internet. An RSU can extend the range of a vehicular ad hoc network by acting as a forwarder. It may also interact with OBUs without the need for Internet connectivity. For example, an RSU that is connected to a traffic light controller could automatically preempt a traffic light when an emergency vehicle enters its range. Aside from creating the opportunity for V2I communication, infrastucture-based networks could also facilitate V2V communication over longer distances. That is, if a vehicle attempts to deliver a message to a region that falls outside of the ad hoc network, RSUs can be used to forward the message to an RSU in a different ad hoc network.

2.1.3 Forwarding strategies

Routing protocols in VANETs control the way entities exchange information. This includes the establishment of a route, data forwarding decisions, and maintaining the route or recovering from failure. Due to the dynaimc nature of VANETs, there is a demand for sophisticated routing protocols. Routing protocols can be divided into two categories: topology-based, and geographic [34]. The Car 2 Car Communication Consortium (C2C-CC) Manifesto [1] describes four types of forwarding strategies that can be used in vehicular communication, as part of the aforementioned routing protocol categories.

Geographical unicast

Geographical unicast (geounicast) can be used to send data to a single, specific destination node. This communication can be through a direct link between the source node and the destination node, or through multiple intermediate hops.

Topologically-scoped broadcast

Topologically-scoped broadcast can be used to send data to all nodes in the scope of the vehicular ad hoc network. Because a naive approach would cause data to propagate for a long time, this broadcast is limited in scope by a maximum number of hops it is allowed to go through.

Geographically-scoped broadcast

Geographically-scoped broadcast (geobroadcast, or geocast) can be used to send data to all nodes within a specified geographic target region. This region is defined with a geometric shape, such as a circle or a polygon. The source node that sends this message can be located inside of this target region. If it is not, geounicast could be used to forward the message to

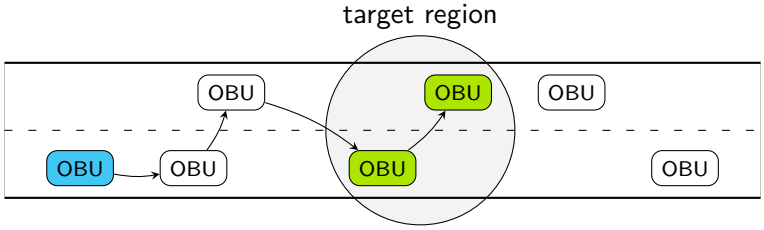


Figure 2.1: Geocasting using an ad hoc network.

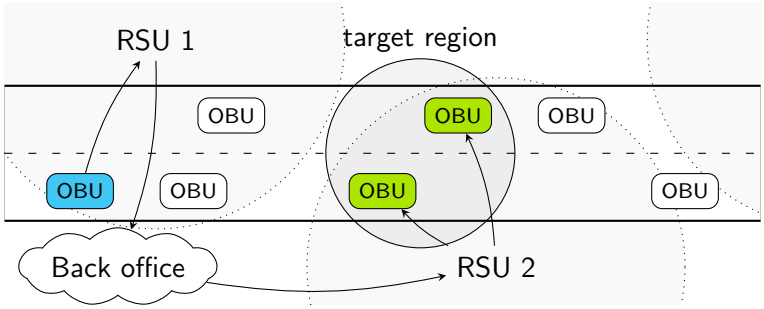


Figure 2.2: Geocasting using a fixed infrastructure.

a node in the target region. An example of a geocast can be found in Figure 2.1. The blue OBU represents the source node, and the green OBUs are the nodes that fall within the target area. The two OBUs in between, and the green OBU on the bottom lane, serve as message forwarders.

Given the same situation, we can see how this scenario would play out in an infrastructure-based network in Figure 2.2. The source node sends its message to the only RSU that it is in range of, RSU 1. Because RSU 1 does not overlap with the target region, the RSU forwards the message to a back office. The back office determines that RSU 2 does overlap with the target area, so it forwards the message to this RSU. Finally, RSU 2 can disseminate the message to the OBUs in the target region.

Geographically-scoped anycast

Geographically-scoped anycast (geoanycast) can be used to send data to any of the nodes within a specified geographic target region. In contrast to a geobroadcast, nodes within the target region will not forward the message to other nodes within the target region. This means that if we would use geoanycast instead of a geocast in the example in Figure 2.1, the green OBU in the bottom lane would not act as a forwarder to the green OBU in the top lane.

2.1.4 Internet-based geocasting

Many geocasting protocols exist [29], but it is not is not desirable to create a new, large infrastructure to realize this functionality. Rather, it would be preferable to integrate geocasting into the existing infrastructure of the Internet and its protocols. This would aid the development of ITS applications, as well as the trend of the Internet of Things (IoT) in general. To facilitate this Internet-based communication network, the concept of a node having a physical location has to be integrated into the logical addressing scheme of the Internet. Several proposals have been made in the literature to solve this problem. We will discuss some of these proposals.

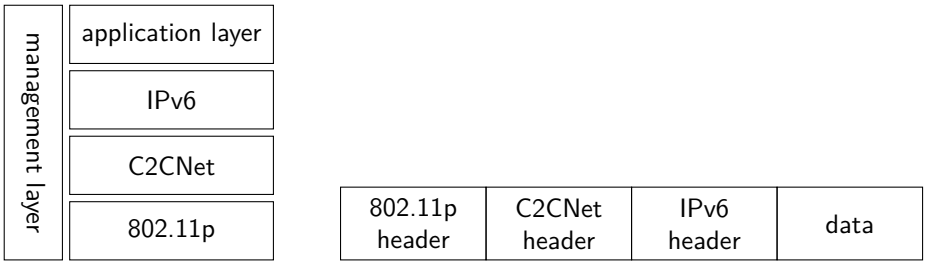
GPS-based addressing and routing

Imieliński and Navas [21, 22] describe GPS-based addressing and routing approaches. Three network components are identified in the geographic routing method: the GeoHost, the GeoRouter and the GeoNode. A GeoHost is software that runs on the client and is responsible for sending and receiving geographical messages. GeoHosts can send messages by building a packet and forwarding it to the next GeoRouter. A GeoRouter is an Internet router that is aware of its own geographical service area, which is the aggregate of nodes that are assigned to it in a hierarchical structure, with the leaf nodes containing small service areas. GeoRouters also exchange the service area information with other GeoRouters, which results in each one being aware of the service area of every other GeoRouter. A geocast message that arrives at a GeoRouter is checked for overlap with the other GeoRouters, and the message is forwarded toward them. If the message overlaps with a GeoRouter's own service area, the message is forwarded to an attached GeoNode. A GeoNode is responsible for distributing the message in the target area. It also buffers messages that have a specified lifetime until the lifetime expires.

The shortcoming of this solution is that a specialized infrastructure needs to be created, because routers need to be aware of service areas.

Geographical IPv6 prefix format

In [15], a global geographic address allocation mechanism based on IPv6 is defined. Compared to IPv4, IPv6 provides a much larger address space of 128 bits. This allows geographic locations to be encoded into the address with high accuracy. The proposal divides the world into squares using a 48-bit field to encode a format prefix, section number, latitude and longitude, offering precision of approximately 6.4 meters at the equator. Bit interleaving is used to support arbitrary granularity. For every bit that the prefix is shorter, the square's coverage area increase twofold. For instance,



(a) Protocol stack (b) Packet encapsulation

Figure 2.3: GeoNet architecture.

a 36-bit prefix can be used to represent a neighborhood, for an area size of approximately 407 meters on a side at the equator.

The solution relies on the allocation of an IPv6 format prefix for this purpose. Without this standardization, the format can not be implemented. In addition, areas can only be represented as square, which may not necessarily be a desired shape for a target area.

GeoNet project

The GeoNet project [8, 27] is a European initiative to create an architecture that combines C2C-CC geonetworking [1] with IPv6 networking to enable both infrastructure-based and infrastructure-less communication. The protocol stack for this architecture is displayed in Figure 2.3a. The application layer runs on top of IPv6, and the IPv6 layer runs on top of C2CNet, which plays the role of the link layer from the IP layer viewpoint. C2CNet runs on top of IEEE 802.11p, but other wireless technologies could also be used. Application data is encapsulated in headers from the different layers, as shown in Figure 2.3b. The IPv6 header contains the destination IP address, the C2CNet header contains the ID of the next hop, and the 802.11p header contains the MAC address of the C2CNet neighbor.

A management layer exists to provide cross-layer functions. The C2CNet layer enables geographic addressing and routing of packets, for which the forwarding strategies discussed in Section 2.1.3 are used. IPv6 multicast is extended to add support for the consideration of geographical areas as an additional scope. GeoNet nodes (OBUs, RSUs) within a specific geographic area (GeoNet domain) can be reached through a single IPv6 virtual link called a IPv6 C2CNet link. Therefore, two OBUs within this area appear to be neighbors to the IPv6 layer, even if they require packets being hopped over multiple OBUs via the C2CNet layer. Beacon messages are used to exchange location information between GeoNet nodes, and this information is stored in a local location table.

In [12], congestion and security are listed as GeoNet issues. [11] also mentions scalability issues as an open problem, as well as the fact that

geocast regions have to be predefined, rather than being able to be chosen by the user.

Extended DNS

In the previously referenced [21, 22], a geographical addressing method based on DNS is drafted as well. A new, specialized top-level domain (see Section 2.2.1) is envisioned (for example, `geo`), with each lower level representing a state, county and finally a sequence of polygon coordinates. This suggestion could be modified to include a country as well. In this case, a query such as the following one could be created, with *`polygon`* being the sequence of polygon coordinates:

```
polygon.University-of-Twente.Enschede.Overijssel.The-
    Netherlands.geo
```

The address would be resolved into IP addresses that are present in the covering geographical area. However, the solution was not discussed further, and the `geo` top-level domain failed to gain approval [20].

In [10], an extension called eDNS[1] is proposed to instead use the existing domain hierarchy combined with a standard to associate geographical locations with these domains. This will be discussed in more detail in Section 2.3, but it is necessary to get an insight into the functionality of DNS first.

2.2 Domain Name System

The DNS is a protocol for name resolution on the Internet [31]. Its most frequently used function is to turn domain names into the IP addresses that are used by devices as identification, as domain names tend to be easier for humans to remember than IP addresses. DNS's function can therefore be compared to using a phone book to find phone numbers based on the name of an individual or company. Aside from mapping IP addresses to domain names, DNS can also be used for associating other information to the domain names. This will be discussed in Section 2.2.2.

Before the introduction of a hierarchical name space, host names were associated with IP addresses in a single file called `HOSTS.TXT`, maintained by the Network Information Center (NIC). This file would then be sent to all hosts using the FTP protocol. This method had several inherent problems. The required network bandwidth would grow quadratically based on the number of hosts, and the number of hosts was indeed increasing rapidly. Organizations that wanted to administer local domain names would also

[1] eDNS is not to be confused with the equivalently abbreviated Enhanced Domain Name Service [37], or the similarly named Extension mechanisms for DNS (EDNS(0)) protocol [45, 5].

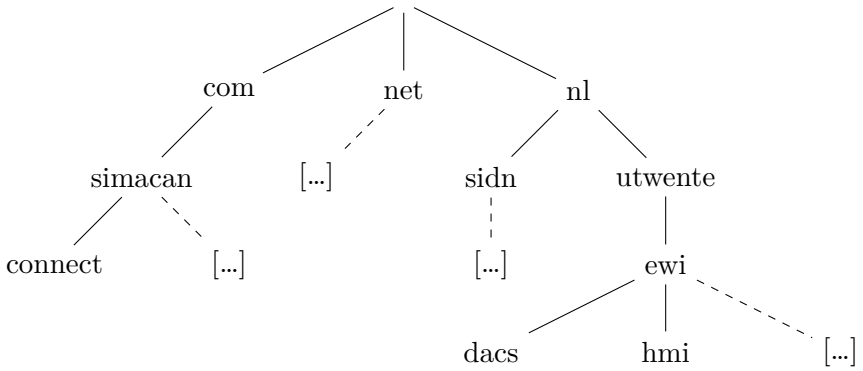


Figure 2.4: Example of a domain name space.

have to wait for the NIC to update the master file before it was usable on the Internet. DNS was proposed as an evolution of other ideas on managing name spaces. It was designed to be hierarchical, to spread operational load, and to delegate administration on name servers.

2.2.1 Domain name space

The domain name space is represented by a tree structure. At the top of the tree is the root node, below this are the top-level domains (TLDs) [35], followed by second-level domains (SLDs) and lower level domains. The two main TLD types are 'generic' (gTLDs) and 'country code' (ccTLDs). Among the more notable gTLDs are `com`, `net` and `org`, while ccTLDs like `nl`, `de` and `uk` represent countries. TLDs are an example of a label, which every node in the tree has. A fully qualified domain name of a node is the list of labels in the tree path up to the root. The hierarchy in a domain name can therefore be interpreted from right to left, as the most specific node is located on the left side.

The domain name space below the root node is divided into sections called zones, created to delegate administrative responsibility to different organizations. A zone starts at a node in the tree, and extends downward until subzones are defined. Therefore, a zone may contain any number of domains and sub-domains (but at least one). The owner of a zone can further delegate anything below the domain name they own.

Although the binary representation of a domain name separates labels with an octet containing the length of the following label, the visual representation separates the labels with the dot ('`.`') character. Because the root node has an empty label, a full domain name ends with a dot to 'separate' the rest of the domain name from this zero-length label. The trailing dot is omitted in most applications. However, when it is not, a domain name ending with the root separator is treated as absolute, as opposed to being

owner	TTL	class	type	RDATA
sidn.nl.	86400	IN	SOA	(ns1.sidn.nl. hostmaster.sidn.nl. 1430295011 4h 1h 1000h 5m)
sidn.nl.	86400	IN	A	213.136.31.216
sidn.nl.	86400	IN	AAAA	2001:7b8:c05::80:1
sidn.nl.	86400	IN	NS	ns1.sidn.nl
sidn.nl.	86400	IN	NS	ns2.sidn.nl
sidn.nl.	86400	IN	NS	ns3.sidn.nl
sidn.nl.	86400	IN	MX	5 kamx.sidn.nl.

Table 2.1: A subset of records in a zone file (based on `http://sidn.nl/`).

relative to a known origin. An example of a domain name space can be found in Figure 2.4. For the `dacs` label, `dacs.ewi.utwente.nl.` would be the fully qualified domain name.

The binary representation of the label may be up to 255 octets, which includes an octet at the start that specifies the length, and a terminating octet that represents the root label. Therefore, the maximum character length of a domain name is 253. A label can be up to 63 octets in length. The labels are allowed to contain the alphanumeric characters (`A` to `Z`, `a` to `z` and `0` to `9`), as well as the hyphen ('`-`') (although it should not start or end with the hyphen). Operations involving the domain name are resolved in a case-insensitive manner.

2.2.2 Resource records

Every zone contains a set of resource information in the form of resource records (RRs). A resource record has the following properties:

owner The domain name this record applies to. An `@` indicates the current domain name.

TTL The time to live of the record, in seconds. This indicates the time for which the record is cached. After this time has expired, new requests will be performed upstream.

class The class of the record. Usually set to `IN` (Internet) for IP routing, but may be set to `CH` (Chaos) or `HS` (Hesiod) as well.

type The identifier for the record that indicates what it should be used for. Common examples will be discussed.

RDATA The data for the record that describes the resource. The content of this depends on the type (and sometimes class) set before.

The resource records are stored in a domain name server's zone file. An example of the data present in such a zone file can be found in Table 2.1 containing five different resource record types. The parentheses in the SOA record are used to group multi-line data. Many resource record types exist, some of them defined in [32]. We will discuss a selection of the most common types here:

A Address record (or host record) in the form of an IPv4 address.

AAAA Address record similar to the A record, but returns an IPv6 address.

CNAME Canonical name record to specify domain name aliases for other domain names.

MX Mail exchanger record for specifying an Simple Mail Transfer Protocol (SMTP) mail server that handles email messages. Includes a numeric value to indicate priority when multiple MX records are defined.

NS Name server record describing which servers contain DNS information for a domain. A domain will typically have at least primary and secondary name server assignments.

SOA Start of a zone of authority that stores the authoritative information of a zone, such as the primary name server of the domain, the hostmaster, a serial number, and values that define how often the data expires and needs to be refreshed.

TXT Text record for storing human-readable information. This record is also occasionally used to store attributes that do not fit in any other record types.

Particularly relevant for our research, is a less commonly used resource record type:

LOC Location record to store a geographical location [6]. The record's RDATA is stored in the following location format:

```
dLat [mLat [sLat]] {`N'|`S'} dLon [mLon [sLon]] {`E'|`
    W'} alt[`m'] [size[`m'] [hp[`m'] [vp[`m']]]]
```

The square brackets (‘`[`’, ’`]`’) indicate optional fields, while the curved brackets (‘`{`’, ‘`}`’) indicate a choice between the fields separated by a pipe character (‘`|`’). The fields between apostrophes (‘`` ` ``’, ‘`'`’) are to be used as literals, the other fields are variable. The variables starting with `d`, `m` and `s` refer to the latitude and longitude's degrees, minutes and seconds

respectively. `alt` refers to altitude, `siz` to diameter size, and `hp` and `vp` to horizontal and vertical precision. These last four variables are specified in meters ('`m`'). The use of the location (LOC) record within eDNS will be explained in Section 2.3.

2.2.3 Name servers

A name server is a server that holds information about a subset of the domain name space and responds to queries made by other name servers and resolvers. It stores the information that it has as resource records in a zone file. The name servers of the root zone of the domain name space, containing a list of the authoritative name servers for all TLDs, are known by resolvers.

Authoritative name servers are name servers that have full information about zones. They will answer queries for domains that fall under these zones. Although an authoritative name server can be capable of answering queries outside of its authority, most authoritative name servers are configured to be authoritative-only.

An authoritative name server can be a master or a slave. The master name server contains the original records in the zone file, and the slaves are updated with copies of the zone file. Multiple strategies for updating the zone file exist. In [31, 32], a polling method is described where the slave queries the SOA record of the master. If the serial number of the master is higher than that of the slave, a full zone transfer is requested (AXFR). [33] defines an incremental zone transfer (IXFR) that only transfers records that have changed. If either the master or the slave does not support the IXFR feature, AXFR will be used instead. Finally, [44] defines a NOTIFY message that a master server can use to inform its slaves that a change may have occurred, after which the slave can attempt an AXFR or IXFR.

Name servers that are not authoritative tend to be caching name servers. Their goal is to save the result of queries in order to reduce the network traffic required for subsequent queries. Caching name servers strongly reduce the load on authoritative name servers, especially higher up in the domain name space tree. If caching name servers were not used, every single DNS query would first have to query the root name servers to find a domain's TLD name servers. Results are cached for the time period specified in the TTL of the relevant resource record in the zone file. Most people use the caching name server configured by their Internet service provider (ISP).

In the example zone file in Table 2.1, we've seen that name servers are defined with a domain name — commonly delegated to a sub-zone of the queried domain. This would result in the problem that according to the NS records, a particular name server should be contacted to learn that same server's address. To fix this circular dependency, the zone contains 'glue' records that provide the IP addresses of the name servers.

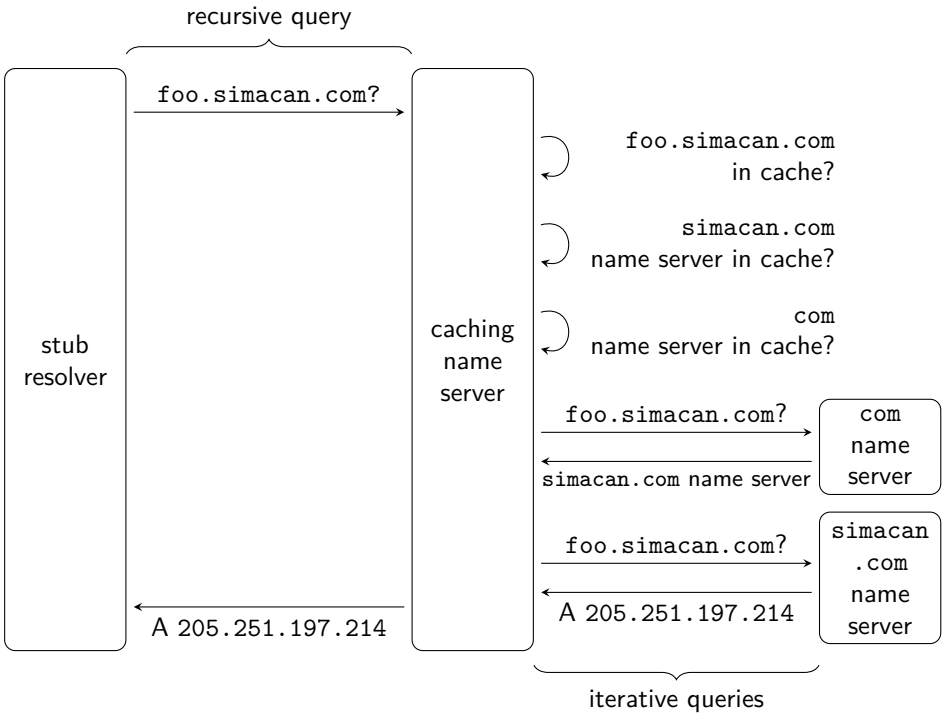


Figure 2.5: Example of the resolution of a DNS query.

2.2.4 Resolvers

A resolver queries name servers based on requests from client programs, and extracts information from the result of these queries. After receiving a query from the resolver, the name server searches for the queried domain in its cache and zone file. If the answer is found, this answer will be returned with a flag to indicate if the data is authoritative. If the name server is authoritative over the requested domain, but the domain name does not exist, the NXDOMAIN error is returned. In the case where the name server is not authoritative over the domain, and the answer cannot be found in its cache, the action to be performed depends on the query type.

An iterative query will result in the return of one or more referrals to name servers that are closer to the requested domain name, allowing the resolver to perform a new query based on this new information. A recursive query will instruct the name server to perform new queries on behalf of the resolver. Name servers are not required to support recursive queries, but the name servers that do are called recursive name servers. Most cache name servers are recursive name servers. Resolvers that do not support referral handling, and therefore rely on recursive name servers, are known as stub resolvers. Resolvers typically maintain their own internal cache as well.

Figure 2.5 shows a scenario of DNS query resolution involving a stub

resolver and name servers. Given the situation where a user tries to access `connect.simacan.com`, the stub resolver on the user's machine performs a recursive query on the configured caching name server. The caching name server does not have `connect.simacan.com` or a name server for `simacan.com` in its cache, but does have a `com` TLD name server entry. Because this name server does not respond to recursive queries, an iterative query is performed. The `com` name server does not know the address of `connect.simacan.com`, but does have an entry for a `simacan.com` name server and returns this referral. The local name server performs an iterative query on the newly found name server, and since this name server is authoritative over the `simacan.com` domain, the requested address for `connect.simacan.com` can be found. The caching name server returns this address to the stub resolver.

2.2.5 Extension mechanisms

The DNS protocol as it was defined in [31, 32] contains restrictions with regards to size that limit the efficient information conveyance in the protocol. EDNS(0) was proposed as a backward compatible way of adding functionality by introducing the OPT pseudo-RR that signals support of specific functionality as metadata [45, 5]. These records are never cached or stored in zone files. Because DNS name servers that do not support EDNS(0) ignore unknown record types, compatibility is preserved.

In addition to adding the OPT RR, several other changes to DNS messages were introduced. Notably, the size limit for User Datagram Protocol (UDP) messages was raised from 512 bytes to 65535 bytes. This was needed to be able to return large data items such as multiple AAAA records for IPv6 compatibility, and DNS Security Extensions (DNSSEC) information to digitally sign zone answers. The introduction of larger UDP packets inadvertently increased the potency of DNS amplification attacks [43]. These attacks will send forged requests to servers, with the source address set to that of the server the attacker wants to target. The servers that receive the request will send their large answers to the target server, thereby flooding the victim.

2.3 Extended Domain Name System

eDNS is an extension to DNS that adds support for querying geographical locations, rather than hostnames. It was first proposed by Fioreze and Heijenk in [10]. Following the proposal, an implementation with delegation was made by the same authors in [11]. The work in [48] improved on this implementation by adding a distributed architecture for more optimal delegation. Most recently, support for dynamic updating was added in [42] to improve

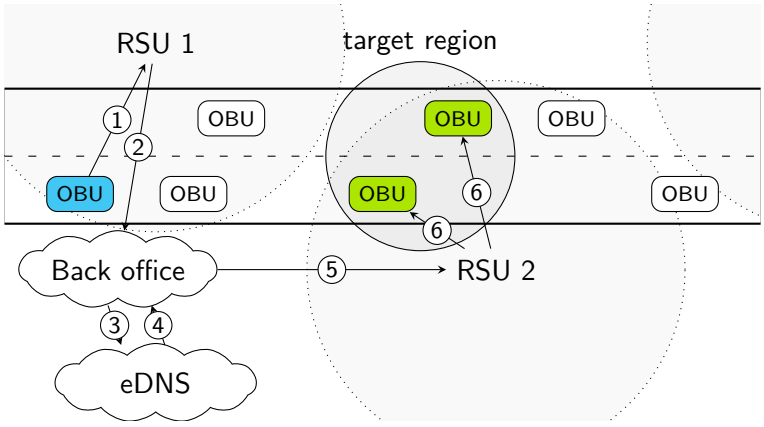


Figure 2.6: Geocasting using a fixed infrastructure with eDNS.

dynamicity management. We will discuss eDNS and all of its improvements in this chapter.

2.3.1 Proposal

As mentioned earlier in Section 2.1.4, Fioreze and Heijenk [10] propose the extension of DNS such that clients can resolve IP addresses based on geographical locations, rather than domain names. The proposal relies on the existing LOC record specification described in Section 2.2.2. The novelty of the proposal is that LOC records are allowed to be used as the primary key for DNS queries, in addition to the methods of using hostnames or IP addresses. The eDNS server would know which records to return by calculating the intersection between geographical locations stored in the LOC records, and the one in a query. The eDNS proposal has various strengths. For one, it is based on the existing DNS architecture, which has proven its high scalability through its use on the Internet. Secondly, it does not require specialized hardware or software, or modification of existing protocols.

The authors foresee eDNS being used only for fixed infrastructure elements such as RSUs. It is argued that storing OBUs may introduce scaling issues due to vehicles' high volume and mobility. The vehicles would have to report their location every time their location changes, which continuously happens during normal operation. A situation similar to the example in Figure 2.2 is envisioned, with the back office querying an eDNS server with the target region, and receiving the IP address of RSU 2. This is illustrated in step 3 and 4 of Figure 2.6.

2.3.2 Methodology

In [11], a prototype implementation is described based on the Name Server Daemon (NSD)[2], an open-source DNS name server program developed by NLnet Labs. The prototype follows the suggestions made in the proposal by adding support for the use of LOC records as the primary key. Here, the LOC records represent RSUs, and the query contains the geocast region. Intersection between geographical locations is calculated by first calculating the distance between the geocast region and the RSUs using the Haversine formula. Then, for each RSU it is checked if this distance is smaller than the combined radii of the geocast region and the RSU. A positive result indicates an intersection.

Geographical queries have the following format:

```
`('dLat mLat sLat {`N'|`S'} dLon mLon sLon {`E'|`W'}
   alt[`m'] size[`m']`)'.domain
```

It follows the format of the LOC record, surrounded by parentheses. It is not clarified of how a LOC record's precision parameters should be interpreted. In addition, minutes and seconds do not appear to be optional, and examples of the new query format omit the use of decimal precision, presumably due to the use of the dot character in a DNS query already being reserved for the separation of labels. The consequence of this is that one can not query a location with sub-second precision. The length of a second depends on one's position on the Earth, but is about 31 meters in the worst case. One may note that the size is described as a radius, while the specification of the LOC record [6] describes the size parameter as the diameter of a sphere.

The geographical query format is hybrid, in the sense that logical domain names can be mixed with a geographical location. The geographical part is used on the lowest level. Thanks to this property, top-level domains are not required to support the geographical format. The document also describes a delegation strategy. A name server that receives a location query will first check if it is authoritative over that region, that is, if its own coverage area intersects with the requested region. If it is not, it will check if it has any referrals to other servers and fully delegates the request to the child name server, which performs the same check. If an intersection does exist, the answer records are returned to the initial name server before being returned to the resolver.

2.3.3 Extended prototype

Westra [48] extended eDNS based on the previously mentioned implementation. The spatial data indexing and search algorithm was improved by using

[2] `https://nlnetlabs.nl/projects/nsd/`

R*-trees, resulting in faster lookup times. A hierarchy that allows for delegation among multiple name servers was also designed and implemented. Lastly, the query format was refined to increase flexibility. Here, we briefly describe this extension.

R*-tree

The extended prototype is based on R*-trees [2] as an efficient spatial data indexing and searching method. R*-trees are a variant of R-trees, which are dynamic, hierarchical data indexing structures, and one of several methods that can be used for spatial data indexing [7]. The leaf nodes in the tree describe several spatial objects surrounded by a Minimum Bounding Rectangle (MBR). The non-leaf nodes similarly group their children together by an MBR. R-trees can be queried by traversing the tree. If the searched point or area overlaps with a node's MBR, that node's children will be checked in the same way until a leaf is encountered, in that case it is added to the result set if it overlaps with the query.

R*-trees differ from R-trees in the way of handling the insertion of new elements in the tree. On creation, R-trees and its variants are configured with a maximum number of entries that can be present in a node. When the nodes in the trees reach their maximum number of entries, they will split into new nodes that each contain at least the configured minimum number of entries. Guttman [14] proposed split algorithms that try to minimize the required enlargement of the bounding box. R*-trees improve on these algorithms by prioritizing the smallest overlapping area, as this reduces the number of paths that need to be traversed. In addition, R*-trees add dynamic reorganization by allowing re-insertions during regular insertion. This attempts to solve the problem that the structure of R-trees is strongly dependent on the order in which objects are inserted. This influences the query efficiency. The improved tree quality comes at the cost of a lower insertion speed.

Delegation

As mentioned earlier, eDNS as implemented in [11] supports basic delegation. It would forward a query to a child server if it was not able to answer the query itself. A shortcoming of this method is that an eDNS server is not aware of the coverage of its child servers. To reduce network traffic, [48] introduces the bounding box (BND) record type. The BND record defines the bounding rectangle of child servers, which prevents blind delegation of a location request if it falls outside of the known coverage of the child server. These BND records are stored as MBRs in an R*-tree, separate from the LOC records R*-tree. The BND record type is implemented as a TXT record type with specifically formatted content, in accordance with [38]. By follow-

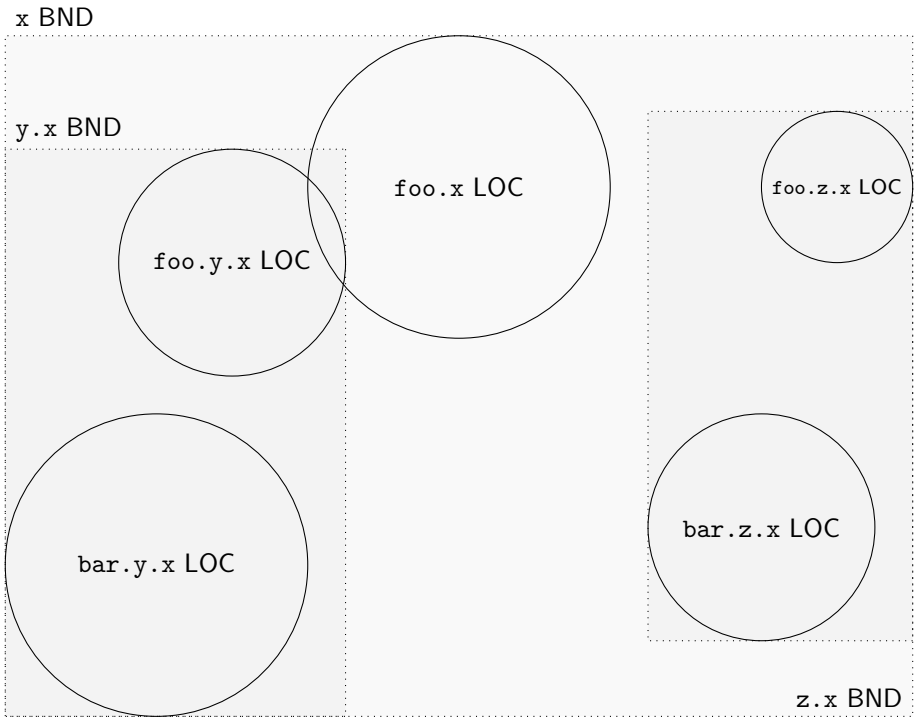


Figure 2.7: Hierarchy of eDNS servers visualized with BND records.

ing this approach, and not modifying the existing NS record to be able to hold a rectangular shape, compatibility with existing DNS implementations is preserved. The format of the BND record is as follows:

```
`v=bnd1' msLatMin msLonMin, msLatMax msLonMax, altMin
   [`cm'], altMax[`cm']
```

The minimum and maximum coordinates, defined in milliseconds, can be used to create a rectangle. The BND records are stored in a separate R*-tree, but should not be saved in the zone file, because they are volatile metadata. DynDNS [47] can be used to keep the BND records in parent name servers updated.

Figure 2.7 shows the hierarchy of three eDNS servers. The BND of `y.x` encompasses `foo.y.x` and `bar.y.x`, as does `z.x` for `foo.z.x` and `bar.z.x`. Although `x` itself only contains a LOC record for `foo.x`, it also encompasses the bounding boxes of the subdomains.

Query format

The query format was changed to improve flexibility. Due to the original format not supporting milliseconds, additional millisecond fields are added to improve the query precision. The altitude and radius size fields are swapped

to be able to make altitude an optional field. These changes result in the following revised format:

```
`('dLat [mLat [sLat [msLat]]] {`N'|`S'} dLon [mLon [
   sLon [msLon]]] {`E'|`W'} size[`m'] [alt[`m']]`)'.
   domain
```

Optional request parameters to reduce bandwidth and processing power can be set as well: a minimum (‘`min`’) and maximum (‘`max`’) record size, as well as a boolean value to indicate if subzones should be searched (‘`sub`’). These parameters have the following format, and may be prepended before a geographical query’s closing parenthesis:

```
[`min='min{`m'|`km'}] [`max='max{`m'|`km'}] [`sub='{`y
   '|`n'}]
```

A format to create polygonal queries with vertices is mentioned as well, but not implemented. Although it would make sense to group the vertex coordinates together, the 63 octet DNS label length limitation makes this unfeasible. Each vertex is therefore contained in its own parentheses.

```
`('coordinate 1`)'.`('…`)'.`('coordinate  `)'.domain
```

2.3.4 Dynamicity management

The previous iterations of eDNS did not explicitly consider dynamic location resource records. They mainly focused on storing the geographic location of nodes that have a fixed location, such as RSUs. In [42], Van Leeuwen extends eDNS with functionality to dynamically manage locations. In the work, it is assumed that a central server exists that tracks dynamics of the environment. Functionality is added to the NSD server (modified for eDNS) that retrieves records from this server, rewrites its own zone file, rebuilds the database, and reloads. Various strategies for determining the refresh rate are discussed and simulated to test I/O and CPU load.

Chapter 3

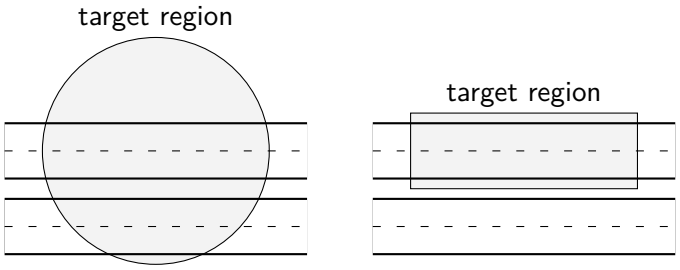
Approach

In Chapter 2, we looked at various technologies that can help us to achieve geocasting within the infrastructure of the Internet. In particular, we have discussed the workings of the eDNS protocol. Although advantages of eDNS compared to its alternatives were identified, there is room for improvements to eDNS to extend its functionality.

One may note that customization of query regions may be of significant importance to many applications. For example, the shape of a highway can be more accurately described with a line than with a circle. Additional ways to define shapes are discussed in Section 3.1. Nearest neighbor querying support can be added to address vehicles that fall outside of the range of RSUs if multi-hop communication is used, as described in Section 3.2. Section 3.3 explains that the addition of an extraction layer can prove beneficial to implement eDNS support in multiple DNS name server software applications. Opportunities for improving the support for dynamicity also exist, in a way that it makes use of the DynDNS method. We consider this in Section 3.4. A prototype of the discussed additions will be made, as expressed in Section 3.5, and further, this prototype will be used in a practical use case. Two potential use cases are explored in Section 3.6.

3.1 Shape definition

One possible area of improvement is geometric shape definitions for querying. Currently, it is possible to issue queries with circular or spherical shapes, and also get the replies back as circular or spherical shapes, as this is how LOC records are defined in the LOC records in DNS zone files. One could imagine targeting a more specific region, such as a long stretch of a road. Using a circular target region for this could result in the message being broadcast to RSUs next to other, irrelevant roads within the circle. This is visualized in Figure 3.1. Figure 3.1a uses a circle to address a stretch of the top road. Using this shape, the bottom road inadvertently falls within the



(a) Circular (b) Rectangular

Figure 3.1: Geocasting to different target region shapes.

target region as well. A rectangular shape, shown in Figure 3.1b, is more accurate for this use case.

The shape limitation was discussed in [48], and the author proposed a way to define polygonal shapes in requests and DNS zone files. However, no concrete implementation of different shapes within eDNS has been done so far. The feasibility of this proposed idea should be examined. In addition to polygons, we can consider the addition of a 'route' shape. It would be generated from a line of connected points surrounded by an area with a specified distance [3]. This shape can accurately describe the real shape of a road using less vertices than a polygon. The rectangular shape shown in Figure 3.1b can be described as a polygon, or approximated with a route shape.

The efficiency of various region shapes within the context of VANETs has been discussed by Jöchle et al. [25]. The research compares circles, rectangles, polygons and routes in various simulated scenarios. The results show that circles generally perform remarkably well in terms of accuracy, and should often be preferred because of their concise specification — a single point and a radius. Nonetheless, the authors note that choosing an appropriate radius is highly scenario dependent.

We intend to add support for the polygon and route shapes to our prototype, in addition to the circle that is already supported by other prototypes.

3.2 Nearest neighbor resolution

eDNS currently only provides a query result if entries fall within the queried area. While this is expected, it could be useful for some applications to provide results that are near the queried area, in case there are no results in the queried area itself. Consider the scenario where one wishes to send a message to an area that is not covered by an RSU. By sending the message to a nearby RSU instead, that RSU could send the message to vehicles within

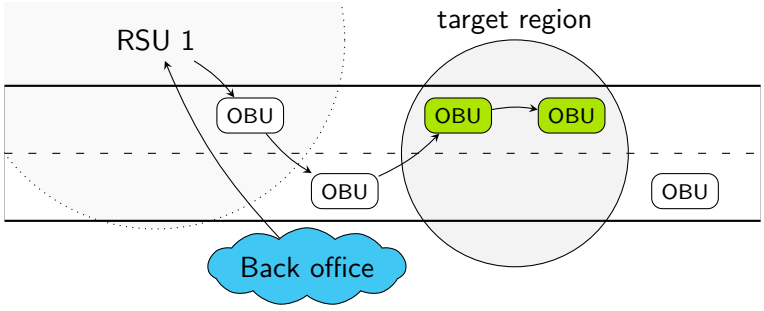


Figure 3.2: Geocasting using nearest neighbor resolution.

its coverage range, and those vehicles could then propagate the message to the intended area through multi-hop communication. The concept is visualized in Figure 3.2. The back office aims to send a message to a target region with no intersecting RSU, but the message can still be delivered by making use of the nearest RSU and nearby OBUs.

The problem of finding nearby nodes in tree data structures is known as k-nearest neighbor (kNN) resolution in the literature, where k is the number of closest results. Solving it for R-trees has been discussed by Roussopoulos et al. [39]. Hjaltason and Samet [18] introduce an incremental approach for finding objects in order of their distance to the queried location, until one is found that meets specified criteria. k cannot be determined beforehand in this case. For example, this could be used to find the nearest RSU that has at least a certain number of vehicles in its coverage range. These vehicles may potentially forward information to the target geocast area in multiple hops. The use of k-nearest neighbor queries in road networks has been discussed by Jensen et al. [24].

A client could request nearest neighbor resolution in an eDNS query by appending a parameter to the geographical coordinates. We propose the following format for this, following the conventions listed in Section 4.2.5: `` `nn='nn ``. For instance, a request for the three locations closest to a specific point may look like this:

```
(1 N 2 E nn=3).simacan.com
```

The authoritative name server should parse this parameter and perform an algorithm to find nearby neighbors, ordered by distance, rather than finding overlapping LOC records.

Nearest neighbor resolution becomes more complicated when delegation is to be considered. Although it is possible for a name server to ask each subdomain about its nearest neighbors, it is not doable to combine results from multiple sources and selecting the nearest neighbors from that set, without knowing the individual distances. One possible solution would be to have the authoritative server request the actual LOC records from the

subdomain, in addition to the record type originally requested (such as A or AAAA). The authoritative server could then parse these records, apply its own distance calculations, and compare these results to the distances of its own LOC records. Another solution is to have the authoritative servers for the subdomains report the distances for its results. Every unique record name would require its distance to be reported. There is no standardized way to transfer such a distance value, so we propose the use of a new volatile record for every unique result name. We will refer to this record as the distance (DST) record. These records would not be added to the persistent zone storage of the authoritative server, but would instead only be generated temporarily for inclusion in the query answer. We propose the following DST record definition as an implementation of the TXT record, using the same formatting rules as in Section 4.2.5:

```
`v=dst1' distance
```

The distance is a decimal value in meters. Because each authoritative server already knows the distances of the requested shape to its own LOC records after doing the local nearest neighbor resolution, no additional computation is required. The parent authoritative server can use these temporary records to order the other result records, and trim the number of records to the requested number. Because DST records are essentially metadata, they are returned in the 'additional' section of DNS query answers. This is comparable to how the OPT pseudo-RR is returned for the EDNS(0) protocol [45, 5].

A considerable drawback of the first proposed solution is that it increases the computational burden on the non-leaf authoritative servers, as they would have to perform additional distance calculations. These distance calculations are wasteful, because they were already performed by the subdomains to return the initial set of nearest neighboring records. After considering this imposed computational overhead, we chose to implement the second proposed solution in the prototype. This does require that all authoritative servers in the tree support this new record type. For this reason, one could implement the first solution as a fallback method for the second solution if an authoritative server detects that a subdomain does not return distances. The DST records may also be returned for non-nearest neighbor queries to aid the main authoritative server in ordering its results by distance. However, because this use case does not require a distance to create a correct resource record set (RRset), providing the DST records remains optional.

Before visualizing the process of nearest neighbor resolution when combined with delegation, let us first look at a regular eDNS delegation process. Figure 3.3 shows the processing of a query directed at the name server `x`. This name server has two subdomains, `y.x` and `z.x`. The client asks `x` for records of a certain TYPE that are located within a requested geographic

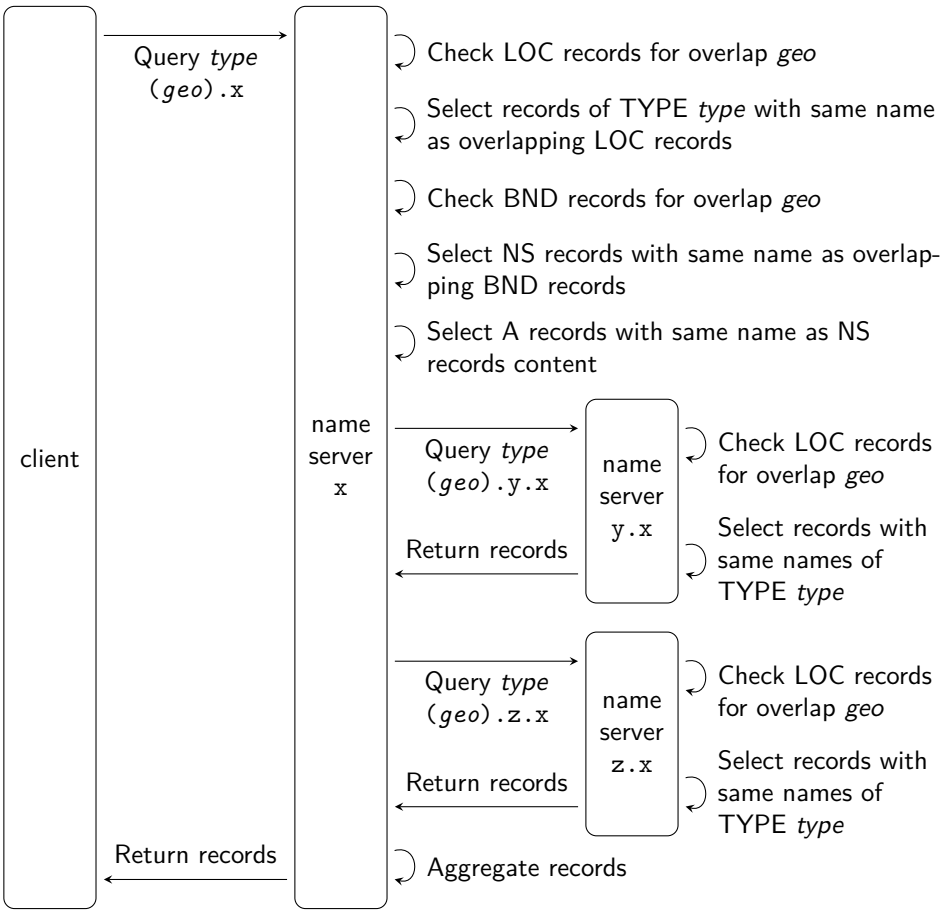


Figure 3.3: eDNS delegation.

region. The `x` name server checks its own LOC records for overlap, as well as the BND records that describe the bounding boxes of the subdomains' LOC records — in our prototype, this is done by querying the synchronized *locations* and the *boundingboxes* tables, respectively. If overlap with the BND record belonging to one or more of the subdomains is found, new queries, directed at `y.x` and/or `z.x`, are created and sent by looking up the NS and its accompanying glue records. `y.x` and `z.x` perform the same process as `x`, but do not have any BND records to check, because they have no subdomains themselves. Any results are returned to the parent name server `x`. `x` then combines them with its own records and returns the final RRset to the client.

A visualization of the processing done for nearest neighbor resolution in a delegated deployment is shown in Figure 3.4. As in the diameter-based delegation example, a query for a certain TYPE of record is sent to a name server `x`. Instead of finding overlapping records, `x` will attempt to find the nearest LOC records. The distances of these records are stored in the form of DST records. Note that unlike the process for normal delegation, no overlap with BND records is checked, because all subdomains have to be queried regardless of the result. Requests are sent to subdomains for ANY records, rather than the type requested by the client, as we would like to receive DST records in addition to the requested type. It is argued by some that ANY queries should be deprecated to prevent their use in amplification attacks [13]. Amplification attacks attempt to overload a victim's bandwidth capacity. DNS ANY queries are well-suited to this type of attack, because the response size of such a request is significantly larger than the request itself. We therefore note that the subdomain requests can alternatively be performed using two separate queries for the *type* and TXT (the base type of DST) resource records.

Assuming `x` and each of its subdomains have at least LOC records, the total number of records of the requested type that are known by `x`, will be . They are typically not included in the records returned to the client, but are used to sort the RRset by distance and trim them to records. These are returned to the client. If less than, or exactly records of the requested type are known, all these records are returned.

3.3 Addition of an abstraction layer

The current implementation of eDNS is based on R*-trees as an underlying data structure. Using an abstraction layer between the DNS name server and the tree structure could offer increased flexibility. This abstraction layer could take care of applying the calculations needed for implementing the mentioned geometric shape definitions and nearest neighbor queries. The use of an abstraction layer could also facilitate the implementation of eDNS

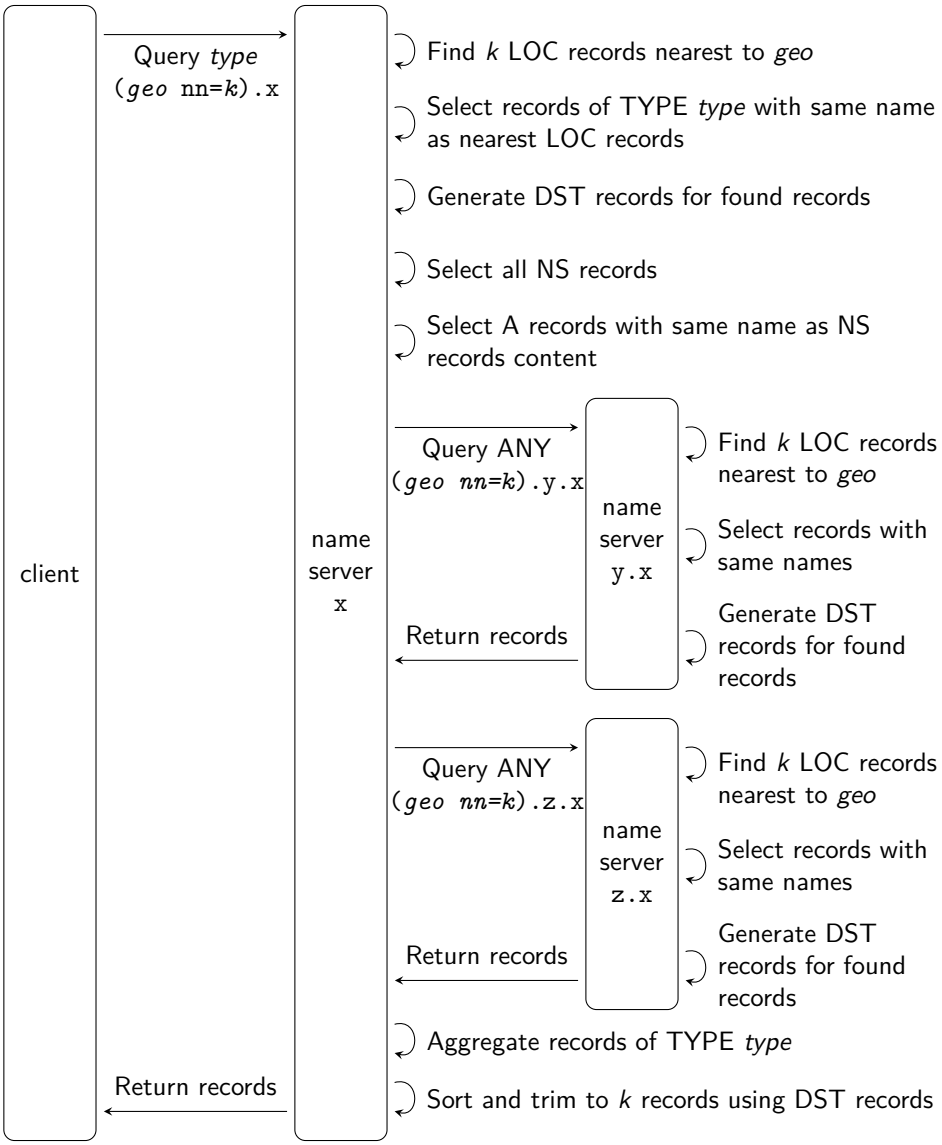


Figure 3.4: eDNS nearest-neighbor resolution with delegation.

in various DNS name server software applications.

One approach is to make use of a database back-end that supports geographical operations. This allows the DNS name server applications to delegate complex spatial operations to this back-end. Furthermore, several DNS name server applications already make use of database back-ends to store resource records [30, Section 2.5.3]. PostGIS[1] is an extension for PostgreSQL[2] databases that adds support for spatial objects in SQL. It has many storage and retrieval options for geographic locations, and can use R-trees as an indexing method — unfortunately, there is no inherent support for R*-trees. Using PostGIS as an abstraction layer, rather than using an R*-tree directly, allows us to take advantage of the advanced features already implemented in PostGIS, such as geographical shape definitions and nearest neighbor queries. These shape definitions are a superset of the 'Simple Features for SQL' objects, defined by the OpenGIS Consortium [16]. PostGIS is reported to have the largest user base of the database-based Geographic Information Systems (GIS) products [41].

3.4 Improved dynamicity

eDNS research so far has mainly focused on nodes with a static location, such as RSUs. The problem of managing dynamic nodes, such as vehicles, has only been discussed in [42]. The author added support for dynamic nodes by implementing a process in the name server software that periodically reads an updated text file with a list of nodes, rewrites its zone file, rebuilds the internal database, and reloads the server. It is noted that an alternative approach to this problem is to use the DynDNS Update protocol [47], but this protocol was not supported by the version of the name server software that was used.

We can implement dynamicity for vehicles using DynDNS Updates by making use of a DNS name server program that supports the DynDNS protocol. A layer of complexity is added when the problem of dynamicity is combined with delegation. An eDNS server needs to know the bounding box coverages of its child servers. If locations in the child servers change, the known bounding boxes may need to be updated as well.

The process is visualized in Figure 3.5. A vehicle `z.y.x` wants to send its new location to name server `y.x` as a DNS Update. `y.x` receives the record and deletes any old `z.y.x` LOC records. Although the client could send a delete DNS Update message along with the addition message to achieve this, we propose that name servers enforce that only one LOC record per name can be present as a way to keep the location data meaningful. `y.x` then inserts new LOC record from `z.y.x`, calculates an updated bounding

[1] `http://postgis.net/`

[2] `http://www.postgresql.org/`

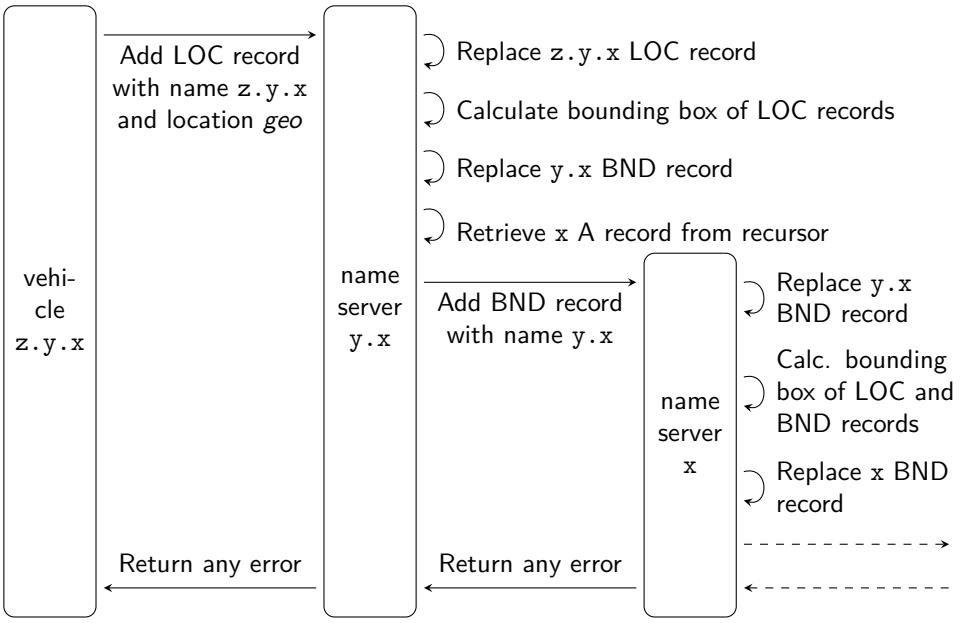


Figure 3.5: eDNS location updating.

box for all its LOC records, inserts the bounding box as a BND record, and sends this as a DNS Update to `x`. We assume here that the inserted LOC record falls outside of `y.x`'s existing coverage or changes the BND record in another way, so that the BND record in its parent name server `x` indeed needs to be updated as well. Similar to the LOC record addition to `y.x`, `x` should remove any old BND records with the `y.x` name to protect the record's meaningfulness, and insert the new record. `x` has to update its own BND record based on its LOC records, and the BND records of its subdomains. This updated BND record needs to be sent to its parent server as well, but because this process is the same as performed in `x`, it is omitted for brevity. If an error occurs at any point, this is sent back to `z.y.x`. Among the possible errors to return are *ServFail* to indicate a server failure, or *NotAuth* if the message is sent to a name server that is not authoritative for `z.y.x`.

A process similar to the one shown in Figure 3.5 is shown in Figure 3.6, and involves the *deletion* of a LOC record. Rather than inserting a new record, only an old entry is removed. Nothing changes from the perspective of `x`, as deletion can also result in the need to send an updated bounding box to the parent server. Finally, one edge case of deletion is shown in Figure 3.7. If the last LOC record in a specific domain is deleted, there is no valid bounding box to be created. The related BND record needs to be removed, both in `y.x` and its parent name server `x`. The latter should

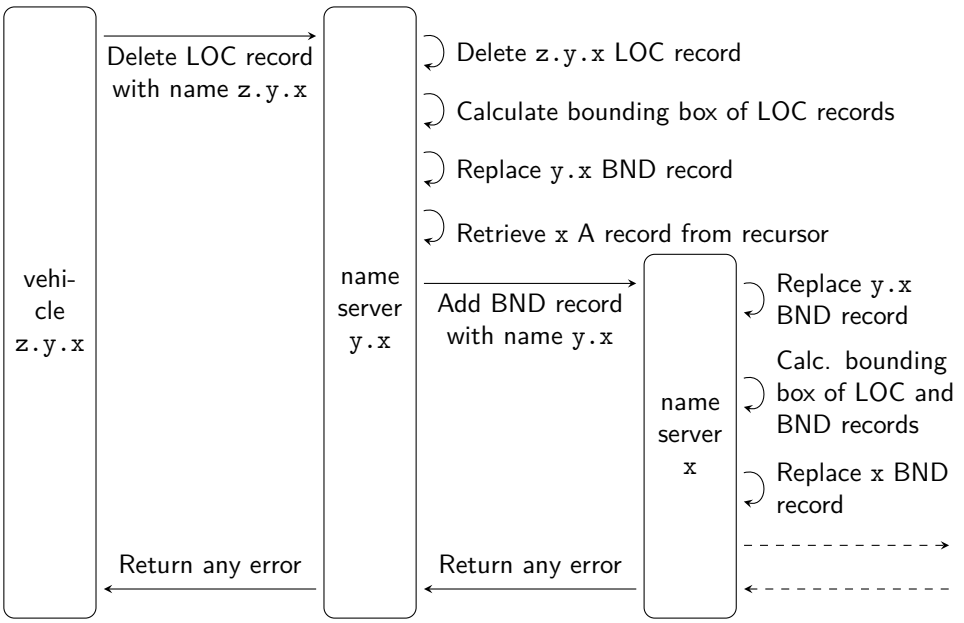


Figure 3.6: eDNS location removal.

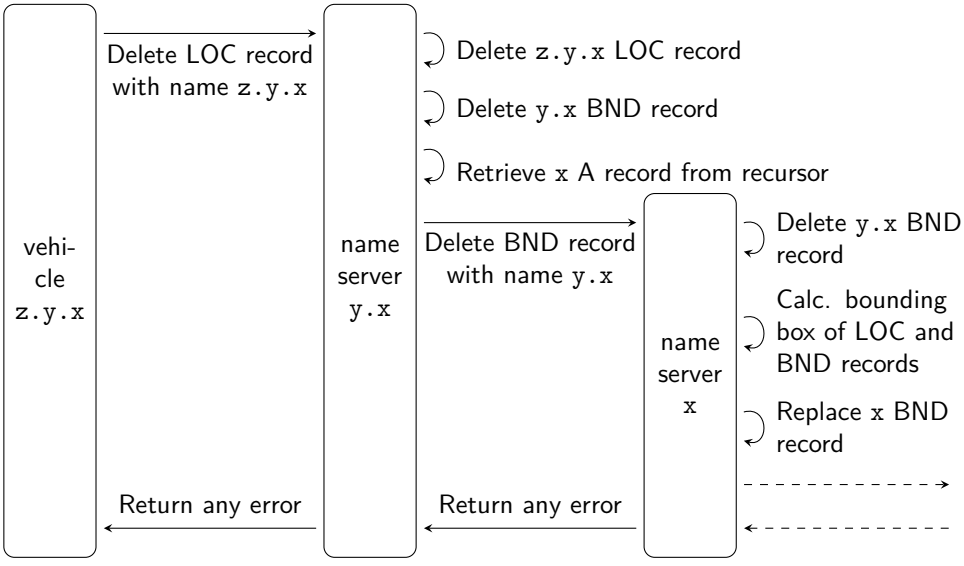


Figure 3.7: eDNS location removal — last location.

therefore be removed with a new DNS Update deletion message.

3.5 Prototyping and validation

A current eDNS implementation exists as an extension of NSD. The extension adds delegation and DynDNS updating functionality for BND records. We consider the use of PostGIS for geographical operations instead. To support the use of a PostGIS database as a back-end, it could be beneficial to focus on different DNS name server applications. As mentioned, DNS name server applications that support database back-ends exist, but NSD is not among them. The open-source PowerDNS[3] natively supports the use of various databases as back-ends, including PostgreSQL. Adding PostGIS support should therefore also be possible. This approach would require the re-implementation of eDNS, as this is not yet supported by PowerDNS.

The previously mentioned improvements should be implemented into the existing DNS architecture. Accordingly, performance should be measured in terms of relevant performance metrics, to be discussed later. We can test the performance by measuring the query and insertion speed of the revised eDNS implementation using geographical data. Data sets containing this geographical data will be based on real world data of RSU or OBU locations, provided by Simacan. Using real geographical data allows us to evaluate realistic use cases more accurately.

3.6 Use case

eDNS is intended to be implemented in a project that Simacan is involved in. Several projects where eDNS could potentially be integrated exist. For example, Simacan contributes to the Spookfiles A58 ('Shockwave Traffic Jams A58')[4] project. This is a project subsidized by the Dutch government that aims to prevent the occurrence of shockwave traffic jams (i.e. traffic jams with no apparent cause). Multiple RSUs have been installed at the A58 highway to assist in fulfilling ITS applications. The locations of these RSUs could be used as targets for eDNS query resolutions.

Another possible use case is the Simacan Control Tower[5] product, which enables companies such as Ahold[6] to follow the position and status of delivery trucks in real-time. eDNS could be used to track the location of the individual trucks. The Dutch program 'Beter Benutten Vervolg' ('Better Utilization Continuation') will invest in the development of user applications for traffic lights. 713 traffic lights (frequently referred to as VRIs) will

[3] `https://www.powerdns.com/`

[4] `http://www.beterbenutten.nl/en/shockwave-traffic-jams`

[5] `https://www.simacan.com/en/products/control-tower/`

[6] `https://www.ahold.com/`

send real-time data of their green, orange and red light timings using the V-Log standard. In addition, 475 traffic lights will be made 'cooperative', which means that they will be equipped with a IEEE 802.11p RSUs to communicate with vehicles. This communication can be used to send the V-Log data gathered from the VRIs to the vehicles, and the vehicles can request priority to the traffic light. A possible extension for the Simacan Control Tower product for the logistics sector, could be to inform trucks about traffic lights status and comunicate with the traffic lights about approaching trucks.

We have chosen to consider the traffic lights use case for our evaluation for two reasons. Firstly, in the context of our research, it is desirable to evaluate the dynamic aspect of eDNS with moving vehicles, rather than with static RSUs. Secondly, it is the most relevant use case for Simacan, because this would be the first use case where Simacan would directly want to address vehicles. To limit the scope of our research, we will not focus on the traffic lights information itself. We will assume that we have a set of vehicles with an associated location. These vehicles need to be informed of certain data, which happens to be traffic lights information for our use case.

Chapter 4

Design

As far as we know, all eDNS prototypes so far have been written as extensions of the NSD DNS name server software. [48] compared NSD to the more established BIND[1] DNS name server program, and concluded that NSD was more suitable to extend due to BIND's inflexibility — reportedly, it includes many internal checks that limit possibilities for extension. In our work, we have chosen to make use of PowerDNS instead. PowerDNS is highly flexible in the sense that it supports a large number of backends (such as databases) to maintain zone information. Additionally, unlike NSD, PowerDNS includes a component that can answer recursive queries, and Dyn-DNS Updates are supported by default. We have extended PowerDNS with the eDNS protocol, by modifying its generic Structured Query Language (SQL) backend.

4.1 Existing architecture

PowerDNS consists of two name server components, an authoritative server and a recursor. The authoritative server only answers queries about domains it knows about, while the recursor can consult authoritative servers to answer queries about other domains. These components are separate, but can be configured to work together, such that an authoritative PowerDNS server may consult the recursor for domains it holds no authority over.

The authoritative server is highly dynamic, as it allows for the configuration of many different backends for storing zone information[2]. The choice of backends includes the classic BIND zone file, MySQL, PostgreSQL, SQLite and Oracle databases, and GeoIP. Some PowerDNS features, such as master/slave configuration and automatic SOA serial generation, are not supported in every backend. The MySQL, PostgreSQL and SQLite backends are the only ones to support all available features.

[1] `https://www.isc.org/downloads/bind/`

[2] `https://doc.powerdns.com/md/authoritative/`

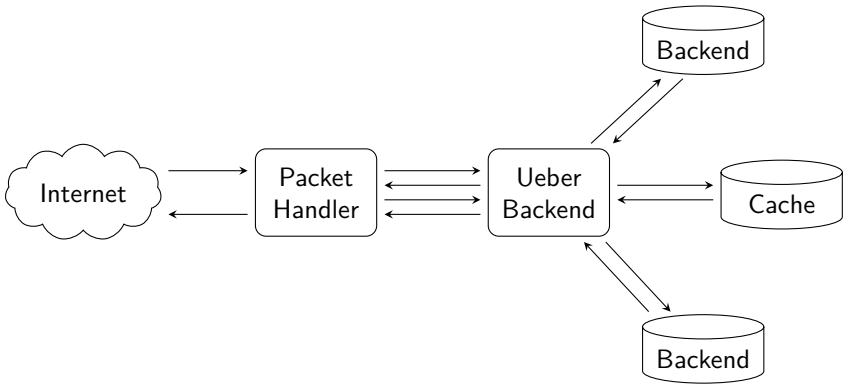


Figure 4.1: PowerDNS packet processing, adapted from [19].

The GeoIP backend[3] allows the mapping of IP addresses to entities such as continents, countries or cities through a mapping table. It is only superficially related to our work, because no specific regions can be queried. It simply maps IP addresses to predefined regions to decide which server should be accessed. This is mainly intended to direct website visitors to a server close to them.

DNS packets that arrive at the authoritative server do not get sent to the backends immediately [19]. Because of the large amount of logic that is needed to answer DNS queries, packets are first processed by the `PacketHandler` class for common operations such as zone selection, wildcard handling and delegation. In its processing algorithm, the `PacketHandler` asks various generic questions to the `UeberBackend` class, which in turn forwards the questions to any number of configured backends, as well as a cache that includes answers of previous questions. These questions include the record lookup for a query name, type and domain, or getting a list of all configured domains. The `PacketHandler` then constructs a result set of records to return to the resolver. A high-level overview of this process is visualized in Figure 4.1.

4.2 Modifications

As mentioned, the SQL based PowerDNS backends have the widest range of supported features. Additionally, some SQL implementations have support for storing geometrical and geographical data. This can simplify our geographical operations. We have therefore extended the existing generic SQL backend, `GSQLBackend`, while adding PostGIS support to the underlying PostgreSQL database to make use of the extensive spatial operations support already present in PostGIS. These operations make use of either

[3] `https://doc.powerdns.com/md/authoritative/backend-geoip/`

`geometry` or `geography` types. The operations that use the `geometry` type perform calculations using cartesian mathematics and straight line vectors. `geography` calculations are performed on a sphere, resembling the shape of the Earth, and are therefore more accurate. The downsides of `geography` are that it is significantly slower, and that not all PostGIS operations support this feature type yet. Accordingly, we have used `geography` where possible to achieve the highest accuracy.

PostGIS objects are supersets of the Simple Features, defined in the Geographic Implementation Specification by the OpenGIS Consortium [17, 16]. This specification lists various geometric object shapes. The three shapes that are important for our research are the Point, LineString and Polygon. No Circle shape is defined. These either have to be represented by a Point object with a separately stored radius, or approximated with the Polygon shape. Each `geometry` object (and `geography` in PostGIS) can be described in a Well-Known Text (WKT) representation [17].

Both the PowerDNS `PacketHandler` and `GSQLBackend` were modified to support the eDNS query format. The `PacketHandler` was changed to not outright reject queries containing parentheses. When a geographical query is detected, the authoritative server performs various additional operations. The geographical labels get parsed into Point objects using the WKT representation. If more than one such Point object is present, they get combined in either a LineString for a line shape, or Polygon for a polygonal shape. Section 4.2.5 details how the choice between these shapes can be specified. Our prototype supports every setting in the defined query format except for the altitude, which remains unprocessed. The reason for this is that at the time of this research, the PostGIS functions that are available for `geography` objects generally do not process the Z component of a coordinate. If support for three-dimensional `geography` objects in PostGIS were to be improved, support for three-dimensional geographical queries in our prototype could be added as well. This would involve parsing the altitude, and passing that value to the Point, LineString and Polygon initialization functions.

In the most basic form of operation, the newly created shape, based on the geographical query, gets checked for overlap with the existing LOC records in the *records* table used by PowerDNS. Checking for overlap between two geographical locations is possible within PostGIS by making use of the `ST_DWithin(geography1, geography2, distance)` function, supplying the combined radii of the locations as the distance argument. However, the LOC records are stored in the *records* table as varchars, not as the `geometry` or `geography` objects required by most PostGIS functions. Therefore, we redundantly store the LOC records in a separate *locations* table in the form of much more computationally efficient `geography` objects. It contains an ID, the fully qualified domain name, a radius and the location itself as a `geography` POINT object. An index is created on the `geography` column to make certain calculations faster. A database trigger is added to

locations	
gid	`serial`
name	`varchar`
radius	`double precision`
geog	`geography(Point)`

(a) locations

boundingboxes	
gid	`serial`
name	`varchar`
geog	`geography(Polygon)`

(b) boundingboxes

locationqueries	
gid	`serial`
qname	`varchar`
geog	`geography(Geometry)`

(c) locationqueries

Table 4.1: New PostgreSQL tables for the prototype.

keep the LOC records and *locations* entries synchronized. The table definition is shown in Table 4.1a. `serial` is a PostgreSQL specific integer type that automatically increments on new inserts and can not be NULL.

If matching LOC records (or their equivalent *locations* entries) are found, the records that that have equivalent fully qualified domain names and are of the requested type and class are returned as a RRset. The prototype results are returned in order of ascending distance, but only if the `no-shuffle` setting is enabled. PowerDNS shuffles its results by default, presumably to aid in load distribution of the results if multiple options are available, similar to the round-robin functionality of the BIND name server software [28, Section 10.7].

4.2.1 Delegation

The problem of delegation for eDNS was solved in [48] by introducing the BND record to store the bounding box of all locations that are stored in an underlying domain. Again, for computational efficiency we have decided to store these bounding boxes in a table separate from the default *records* table: *boundingboxes*. It contains an ID, the fully qualified domain name, and the bounding box as a Polygon `geography` object, along with an index on the `geography` column. This allows us to check for overlap using the same `ST_DWithin` function used earlier. The table definition is shown in Table 4.1b.

All bounding box matches will have their accompanying NS records returned to the `PacketHandler`. The `PacketHandler` can use these NS records to contact subdomains. A new geographical query is generated with the same geographical part, but appended to the subdomain to make sure the subdomain considers itself authoritative over the query. For instance,

`(geo).x` will be turned into `(geo).y.x`, where `y` is the subdomain of `x`. The IP address of the subdomains can be found from the glue records defined in the parent domain.

The connection to the subdomain name servers is set up using a UDP socket. A Transmission Control Protocol (TCP) socket could be used instead for higher reliability at the cost of latency. It would also work around the default maximum UDP packet size of 1680 used by the PowerDNS, which could be insufficient if the query contains many results. Although the UDP packet size threshold can be increased, high values are not desirable because they increase the risk of amplification attacks [43]. The `PacketHandler` waits for the resulting RRs returned by the subdomain, and merges them into the non-delegated results. Only one level of delegation is discussed here, but the servers that hold authority over subdomains can employ the same delegation mechanism themselves.

The parent name server directly queries its subdomains, but a mechanism where other name servers need to be reached can be envisioned, in which case a recursor can be employed for the query resolution. Although not used for this purpose in the prototype, the PowerDNS recursor was modified to support this use case. The PowerDNS recursor would initially reject results that had a name that was different from the query name. This is a problem for eDNS queries, as a query of the form `(geo).x` will yield results with regular fully qualified domain names such as `z.x`, not `(geo).x`. This limitation was therefore removed.

4.2.2 Nearest neighbor resolution

Support has been added to get a specific (maximum) number of results that are nearest to a given geographical shape. This is known as nearest neighbor resolution. PostGIS provides the `ST_Distance(geography1, geography2)` function to get the distance between `geography` objects, but this is fairly slow when applied to all objects. The `<->` centroid distance operator can be used instead to get the distance between the center of the bounding boxes of two `geometry` objects, which makes efficient use of the index created on the `geography` column. However, this is imprecise because the center of the bounding box may not accurately represent the shape. We therefore employ a hybrid method that makes a larger pre-selection based on the centroid distance operator. The `geography` objects are cast to `geometry` for this. The final selection is made by running `ST_Distance` over the approximated pre-selection. This method is also described in [36].

CLASS	TYPE	RDATA	operation
zone CLASS	*RRset TYPE*	*RR RDATA*	add to RRset
ANY	*RRset TYPE*	*empty*	delete RRset
ANY	ANY	*empty*	delete all RRsets from name
NONE	*RRset TYPE*	*RR RDATA*	delete RR from RRset

Table 4.2: The mode of operation based on values in the DNS Update message.

4.2.3 Dynamicity

The version of PowerDNS that our prototype was developed on, version 3.4, contains experimental support for DynDNS[4]. This allows us to simplify the process of managing dynamic records. [47] defines a DNS Update message as an extension of the regular DNS message format defined in [32]. Two modes of operation are offered in this new message: addition and deletion. Using the `GSQLBackend` backend, PowerDNS will insert an entry in the *records* table upon receiving an update message request to add a RR, and delete one or more entries in this table for deletion requests. Both operations make use of the same Update message, but the exact way the message is processed depends on the supplied CLASS, TYPE, and RDATA fields, as can be seen in Table 4.2. The `nsupdate` utility can be used to request DNS updates.

Allowing DNS Update messages to come from any source can be highly undesirable from a security perspective. Two possibilities exist to restrict access. A range of IP addresses may be configured that are allowed to perform updates, where updates from any IP address that falls outside of this range will be rejected. This can be configured globally, or per zone for more granularity. The second option is to generate a TSIG [46] key using one of various authentication algorithms. This key should be known to PowerDNS and supplied as a parameter to DNS Update messages. This guarantees that only authorized clients can perform updates.

The prototype adds a database trigger that activates on the insertion of new location. A new BND record is created that represents the bounding box of all locations for the relevant domain. The `ST_Extent(geomfield)` operation is used for this. This updated coverage should be made known to the parent name server, if the coverage has changed and the parent name server is eDNS-aware. In a realistic use case, a DNS recursor such as the PowerDNS recursor should be used to retrieve the parent domain's IP address. However, that mode of operation is not applicable in our testing setup because we are dealing with fictional domain names. We have therefore added a new *PARENT-ADDRESS* entry in the existing PowerDNS *domainmetadata* table to configure the IP address of this parent name server.

[4] `https://doc.powerdns.com/md/authoritative/dnsupdate/`

The prototype will attempt to send the updated BND record to the parent name server through the same DNS Update mechanism as described before. This required changes to the PowerDNS `PacketHandler` class.

In the context of DNS, an empty non-terminal (ENT) is a domain that owns no resource records, but has one or more subdomains that do [47]. For instance, the domains `x` and `z.y.x` might have resource records, while `y.x` might not. If `z.y.x` were to be deleted, the existence of `y.x` would be redundant. PowerDNS ensures ENTs are inserted and removed as needed upon processing DNS Update packets. A problem arises for the PostgreSQL backend when this procedure is executed on large tables. The SQL command to search for matching domain names uses the `LIKE` operator with a leading wildcard. This operation can not make use of the existing B-tree index[5] on the relevant table column, which significantly reduces the DNS Update performance. For our prototype, we have disabled the ENT related functionality, because it is not relevant for our use case. However, it is likely that indexes can be created that enable faster lookups. The trigram based pg_trgm[6] PostgreSQL module includes index operator classes that are potentially suited for such operations. Alternatively, domain names could be reversed before insertion into the database, which would enable trailing (rather than leading) wildcards for efficient B-tree index searches with the `LIKE` operator.

4.2.4 Visualization

A logging option has been added to the prototype for debugging purposes. This is enabled by adding `` `log='{`y'|'n'} `` as a parameter to the geographical query, similar to how the `nn` and `close` parameters are defined in Section 4.2.5. Because this is purely a debugging option, it is not listed in Section 4.2.5 itself. Upon receiving a geographical query with this parameter enabled, the prototype will write the query and its `geography` to a *locationqueries* table (definition shown in Table 4.1c). `ST_Buffer(geography, radius_of_buffer)` is used to give the `geography` object a surrounding buffer.

The table can be used as a data source in a graphical tool such as QGIS[7]. An example of how the stored data can be visualized is shown in Figure 4.2. The circular shapes are shown in blue, the polygonal shapes in green, and the line shape in orange. They are rendered based on the content of the *locations* table. The visual output can be used to check whether the results for a given geographical query are correct.

[5] `http://www.postgresql.org/docs/current/static/indexes-types.html`
[6] `http://www.postgresql.org/docs/current/static/pgtrgm.html`
[7] `http://www.qgis.org/`

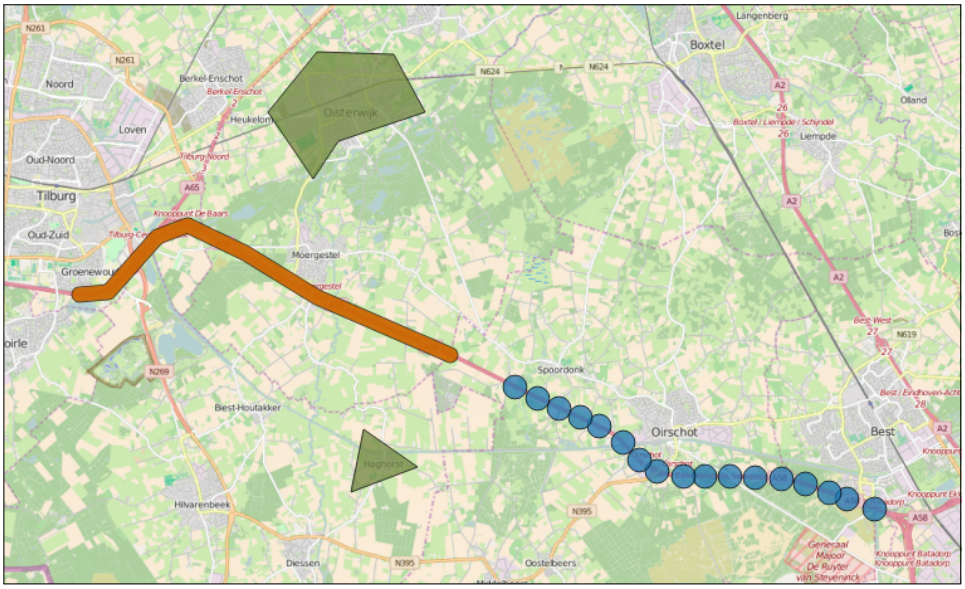


Figure 4.2: Query visualization in QGIS.

4.2.5 Query format

Definition

Several refinements have been made to the geographical query format defined in earlier research. This has resulted in the following label format definition:

```
(d1 [m1 [s1]] {`N'|`S'} d2 [m2 [s2]] {`E'|`S'} [siz[{`
    m'|`km'}]] [alt[{`m'|`km'}] [`nn='nn] [`close='{`y
    '|`n'}])
```

Here, the square brackets (‘`[`’, ’`]`’) indicate optional fields, while the curved brackets (‘`{`’, ‘`}`’) indicate a choice between the fields separated by a pipe character (‘`|`’). The fields between apostrophes (‘`` ` ``’, ‘`'`’) are to be used as literals, the other fields are variable. The meaning of the variables is as follows:

d1, m1, s1	Latitude in degrees [0–90], minutes [0–59.999…], seconds [0–59.999…] (decimal)
d2, m2, s2	Longitude in degrees [0–180], minutes [0–59.999…], seconds [0–59.999…] (decimal)
siz	Size as diameter [0–90000000] (decimal)
alt	Altitude [-100000–42849672.95] (decimal)
nn	Amount of nearest neighbors to return, or 0 (default) to disable nearest neighbor search (integer)

The literals are defined in the following way:

'N', 'S', 'E', 'W'	North, south, east and west; south and west result in a negative latitude and longitude, respectively
'm', 'km'	Meter (default) and kilometer
'y', 'n'	Yes and no
'nn='	Setter to use nearest neighbor search instead of the default intersection search
'close='	Setter to use a polygonal shape instead of the default line shape in case of at least three geographical labels

Due to constraints in the allowed characters in a label, the dot ('`.`') character can not be used as a separator in decimals. Therefore, the underscore ('`_`') character must be used as a substitute, or the fractional part of the decimal can be omitted entirely.

A single geographical label represents a point value with an optional circular buffer of size `siz`. To create more complex shapes, multiple geographical labels can be chained. Two geographical labels always create a line shape. Three or more geographical labels create a line shape by default as well, but can be modified to create a polygonal shape instead by correctly setting the `close` parameter. This setting will cause the last coordinates to automatically loop back to the first coordinates to close the polygon. Parameters can be set in any of the geographical labels, but parameters in labels further to the right take precedence. Unrecognized parameters are ignored.

Modifications

The geographical query format is based on the LOC record format, but it has been heavily modified during several eDNS iterations. Let us summarize the total set of differences of our eDNS iteration compared to the original LOC record format.

- Support for decimals in degrees and minutes has been added.
- Support for setting the size and altitude in kilometers has been added.
- Support for setting horizontal and vertical precision has been removed because they serve no use in our extension.
- Support for additional parameters has been added; `nn` and `close` in our implementation.
- The size and altitude fields have been swapped to allow the size to be set without having to specify an altitude.

- The default size has been changed from one meter to zero.
- The use of parentheses to surround the definition has been made required to differentiate it from a regular domain label.
- Dots are substituted for underscores in decimals.

Note that in comparison to earlier eDNS specifications and implementations, this eDNS specification reverts the size field to represent a diameter as it is used in LOC records, rather than the radius definition mentioned in [11] and [48].

Examples

Based on the preceding specification, examples of valid eDNS queries can be listed:

Simple
`(0 N 0 E).simacan.com`

Decimals
`(1 2 3_4 S 5 6 7_8 W).simacan.com`

Size
`(7 S 8 E _9km).simacan.com`
900 meter diameter, not radius

Nearest neighbor
`(42 N 42 W nn=5).simacan.com`
5 nearest neighbors

Line
`(9 0 N 0 1 E).(9 0 S 0 1 W).simacan.com`

Polygon
`(4 N 8 E).(15 N 16 E).(23 S 42 E close=y).simacan.com`

Complex
`(51_5507 N 5_1252 E).(51_5504 N 5_1375 E).(51_5025 N 5_2926 E).(51_4938 N 5_3094 E).(51_4905 N 5_3640 E).(51_4945 N 5_3874 E).(51_4841 N 5_3923 E 100m close=n).simacan.com`
`close` is off by default, so setting it to `n` is optional

Chapter 5

Evaluation

In this section, the developed prototype is evaluated in terms of most relevant and illustrative performance indicators. We test the implemented functionalities with various configurations and settings to get insight into system behavior in terms of throughput and latency. The evaluation is performed with sets of real historic vehicle location data to simulate the realistic use case of tracking vehicles in an eDNS system.

5.1 Test setup

The open-source Docker project[1] is used to simplify the deployment of multiple instances of the prototype. Docker allows applications to be packaged and run inside isolated, virtualized containers. Each name server can be represented by the combination of a PowerDNS and a PostGIS container, both of them automatically configured through the use of a script file known as a Dockerfile. Figure 5.1 visualizes a possible deployment of the prototype, given the fictional domain `x` with subdomains `y.x` and `z.x`. The PowerDNS instances are linked to their respective PostGIS instances, allowing them to access the database as if they were running in the same environment.

The prototype is deployed to Amazon Web Services (AWS) for reliable performance testing. Amazon allows customers to run code through their EC2 Container Service (ECS) platform. ECS is a layer on top of Elastic Compute Cloud (EC2) that provides the ability to run Docker containers on EC2 instance clusters. Multiple instances of PowerDNS are deployed to ECS clusters. Although it would be possible to run a PostGIS Docker container inside each of these clusters as well, we have opted to make use of AWS's dedicated database platform, the Relational Database Service (RDS). Using RDS allows us to make use of its database management features, which include simplified launching and deletion of databases, and making snapshots of database states. A PostgreSQL database instance is launched for

[1] `https://www.docker.com/`

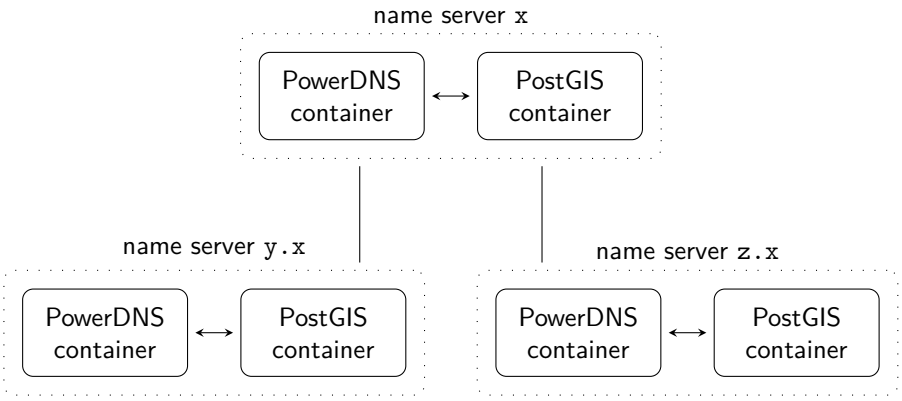


Figure 5.1: Deployment of the prototype using Docker.

each PowerDNS EC2 instance, and all database instances are extended with PostGIS functionality using the `CREATE EXTENSION postgis;` command. Within this setup, PowerDNS instances can make use of one shared RDS instance, but multiple instances are used to prevent the eDNS servers' load from influencing the performance of the other servers. All instances are physically running in the same region (Ireland, `eu-west-1`), the same availability zone within that region (`eu-west-1c`), and are allocated to the same Virtual Private Cloud (VPC).

Both the EC2 and RDS instances are of the type *m3.medium*, which has one virtual CPU on an Intel Xeon E5-2670 v2 processor running at a clockspeed of 2.5 GHz. The amount of available memory per instance is 3.75 gigabyte. Exact specifications for the network performance are not mentioned, but the performance is described as being 'moderate'. This type of instance is, at the time of evaluation, the lowest-cost solution available from Amazon that has consistent performance, and should therefore make the results easily reproducible. *t2.micro* instances are significantly cheaper, but do not guarantee consistent performance – instances of the *t2* type occasionally burst CPU performance above the baseline[2]. Experiments that attempt to replicate or compare to our results should be run on instances with equivalent computing performances of 3 EC2 Compute Units (ECU) each.

All communication with the prototype deployed in AWS is performed from a server in The Netherlands with a bidirectional connection speed of 1 Gb/s.

[2] `https://aws.amazon.com/ec2/instance-types/#burst`

5.2 Measurements

Given the focus of our work, it is important to evaluate the performance of the system from the following perspectives:

- Querying nodes within a specified geographical region.
- Querying nodes with the smallest distance to a specified geographical region.
- Updating nodes with dynamic geographical locations.

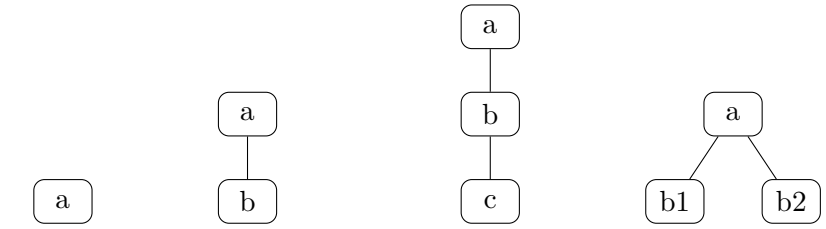
Location data is needed to evaluate these perspectives. We have used a historic location data set of real vehicles. This data set contains frequent updates of vehicles over a period of one month. The frequency of location updates in the data set depends on the number of vehicles that are active, which is dependent on the time of day. This averages to 0.77 unique location updates per second, but can roughly double during the rush hours to 1.42 updates per second. The highest observed number of updates within one minute was 218, bringing the maximum expected update rate to 3.63 location updates per second. In total, close to 1000 vehicles are active during the month that is represented in the data set. Due to the nature of the data, the locations are not uniformly distributed over the total covering area. Updates are most likely to come from highways, as well as origins and destinations shared by the vehicles. In addition, more vehicles are active in densely populated areas. This realistic set of data allows us to evaluate the suitability of our eDNS implementation for use in the Simacan use case described in Section 3.6. A location will be represented by a LOC record, as well as an A record with a fictional IP address belonging to the vehicle.

5.2.1 Performance metrics

The performance of the system can be quantified in the form of throughput and latency. For our use case, the throughput determines how many vehicles the system is able to keep track of. The latency influences the responsiveness of the system, and how up-to-date the stored locations are. The chosen metrics can be defined as follows.

Throughput
: The throughput describes the rate at which the system can process data. In the context of our work, it is the number of queries and also the number of location record updates the system can handle in one second.

Latency
: In our system this metric specifies the time interval between issuing queries and receiving the corresponding reply back.



(a) Minimal (b) Simple (c) Scaled depth (d) Scaled width

Figure 5.2: Evaluation server setups.

(a) 100 locations (b) 1000 locations (c) 10000 locations

Figure 5.3: Data set subsets.

5.2.2 Input parameters

We test the influence of server setups by configuring multiple name space tree setups. Various testing setups are visualized in Figure 5.2. The setups provide a variety in terms of tree width and depth. The total number of nodes for each setup is limited to 3 to limit expenses. Query performance will be tested by querying the top node in the name space tree. The LOC records are divided over the edge nodes. The allocation of the records to the edge nodes will be close to optimal, in the sense that each edge node will be responsible for its own geographical region. This causes the bounding boxes of the edge nodes to be relatively small, with no overlap. The performance of location updates will be tested by sending DynDNS updates to one or more of the available edge nodes and measuring the performance in terms of throughput and latency.

Various input parameters of the setup can influence system performance, as introduced below.

LOC record count

The number of LOC records present in the system. This can be evaluated with a varying number of LOC RR inserted in the eDNS servers.

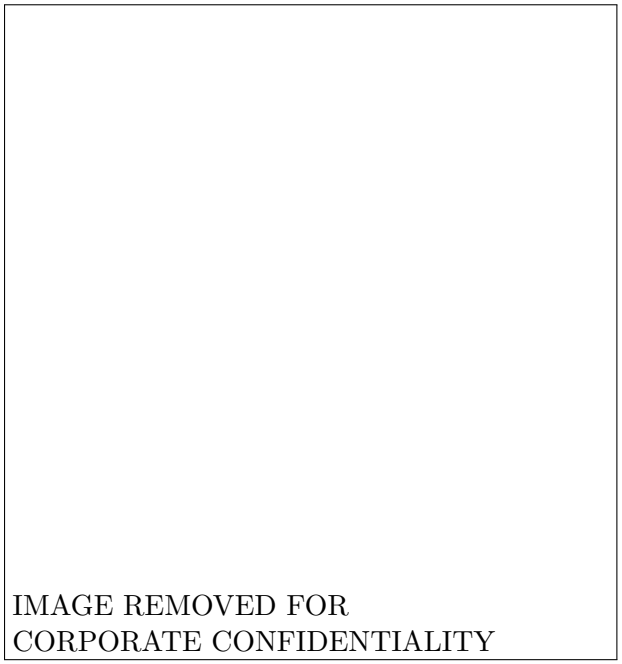


Figure 5.4: Server setup (d) location allocation.

We have taken subsets of various sizes from the previously mentioned data set to evaluate the influence of the amount of LOC records in the databases. Three subsets containing 100, 1000 and 10000 locations were created. A visualization of the subsets is shown in Figure 5.3. Since each of the locations represents a vehicle, the diameter of the corresponding LOC record should be small. A LOC record can not accurately represent the shape of a vehicle, but we have chosen the default LOC diameter of 1 meter as the diameter for each record. As mentioned earlier, multiple setups will be tested, and records will be added to one or more edge nodes based on their geographical position. An example of how a setup's bounding boxes could look is shown in Figure 5.4, demonstrating the subset of 1000 locations divided over setup (d)'s two edge nodes. Each edge node is allocated roughly the same number of LOC records. Upon receiving a DynDNS update message that inserts or removes a LOC or BND resource record, the BND record of the node itself will need to be renewed. The total number of LOC and BND servers can therefore influence the performance as they increase the number of spatial database operations that need to be performed.

Queries are also influenced by the following input parameter in case of diameter-based queries.

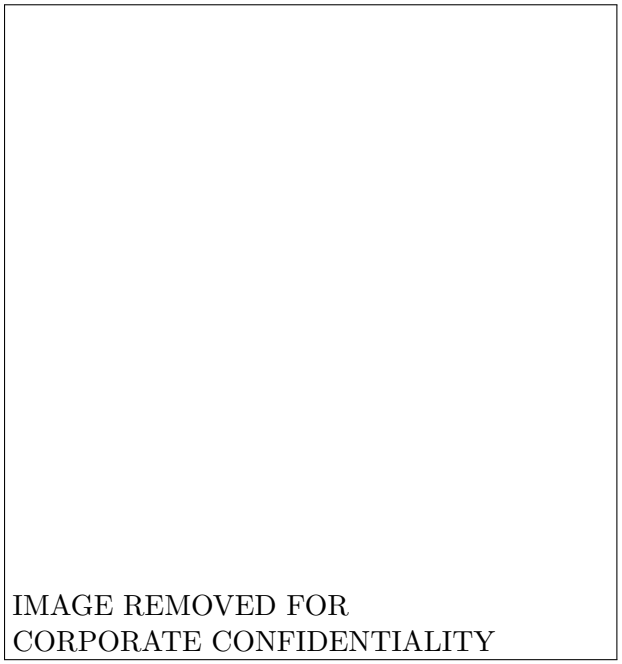


Figure 5.5: Query location distribution.

Query location diameter

The size of the queried circular area, described with a diameter. It can be tested on the same data set by querying various area sizes. For this, we can consider querying areas with diameter sizes as low as 1 meter up to 500 kilometer. The larger sizes in this range would encompass our entire data set. The query locations originate from the same data set as the locations described earlier, but represent a different subset. They do share the same characteristics of being more likely to refer to a location on a highway or a shared origin or destination. The set of 5000 locations used for query evaluation can be seen in Figure 5.5, with each of the query areas visualized with a diameter of roughly 10 kilometers. Higher color opacity represents a larger query density.

Nearest neighbor queries are instead influenced by the amount of requested results.

Query nearest neighbor count

The number of requested nearby results. It can be tested with simple integers. We have chosen to vary the number of requested results from 1 to roughly 500. The same location data set as described in 'Query location diameter' will be used.

5.2.3 Other considerations

All DNS query and update operations are performed with a reasonable timeout of 1 second. If no result is received before this timeout expires, the operation is considered to have failed, and will not contribute to the overall throughput.

In common situations, combining large data sets with large query areas will result in a large number of matching locations. If we were to return all results, we might encounter packet size limitations that prevent us from doing so. DNS packets over TCP are limited to 65535 bytes because of a 2 octet size header, while UDP, which we use for our queries and updates, was traditionally limited to 512 bytes [31, 32] and extended to 65535 bytes by EDNS(0) [5]. However, as mentioned in Section 4.2.1, this maximum UDP packet size is commonly limited to lower values because of the risk of amplification attacks. The 4096 byte limit recommended in [5] is often used instead.

In our testing setup, each PowerDNS eDNS server connects to a database backend on another server. The latency between these servers can be measured to determine its influence on the other results. The `ping` command was used from an EC2 instance to determine the communication delay between two servers in the same availability zone. Executing 20000 consecutive `ping` requests showed that latency between servers in the same availability zone lies within the range milliseconds with .

5.3 Numerical results

For the sake of simplicity, we assume that the throughput and latency data is normally distributed. This allows us to compute confidence intervals for the data points. Performance in the graphs is displayed with a 95% confidence interval ( ). Note that some graphs contain confidence intervals that are too small to be visible.

In order to get reliable results for throughput of the system, it is necessary to have reasonable packet arrival rate such that the system does not wait for packets to arrive. On the other hand, the system should also not be overloaded to the point where it is unable to reduce its internal buffer or respond within timeout limits. To prevent the system from idling, it is necessary to send multiple packets simultaneously. Initial tests have been performed to evaluate the performance with different numbers of simultaneous packets. This is implemented as a thread pool, with each packet being sent in its own thread (a 'worker'). The results of evaluating different number of workers for various combinations of query diameter and data set size can be seen in Figure 5.6. As expected, employing more workers generally results in higher throughput, although with significant diminishing returns. For every tested configuration, throughput converges to a certain

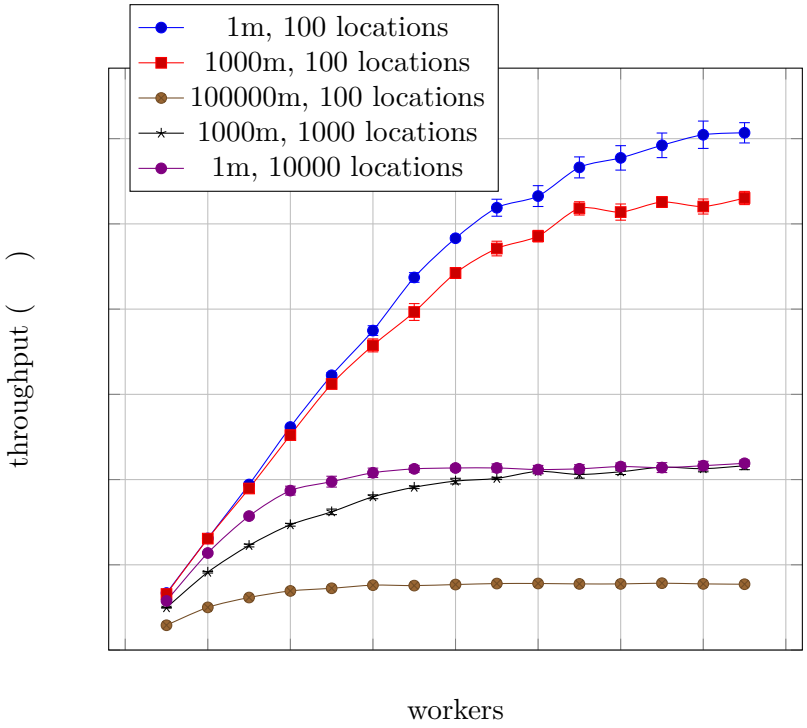


Figure 5.6: Influence of concurrency on throughput (setup (a)).

rate where at least one non-buffer related performance bottleneck emerged, such as limited processor power.

Even though higher number of workers do not negatively affect throughput performance, we can see in Figure 5.7 that using an arbitrarily large number of workers is not reasonable because of latency. Latency appears to consistently increase when a larger number of workers is used. This happens because the server has to divide its resources over all incoming requests, resulting in longer processing times for each request. Based on these results, we do not consider there to be an optimal number of workers, as even for specific configurations it is a trade-off between throughput and latency. We have opted to perform the rest of the evaluation with 8 concurrent workers. At that number of workers, the majority of the potential performance increase from concurrency has been achieved under most testing configurations while having lower latencies than any higher number of workers.

5.3.1 Querying

Throughput and latency have been evaluated for every combination of setup (shown in Figure 5.2), data set size (shown in Figure 5.3) and diameter. The throughput and latency of the system with setup (a) are shown in Figure 5.8 and Figure 5.9, respectively. One may note that both the query diameter

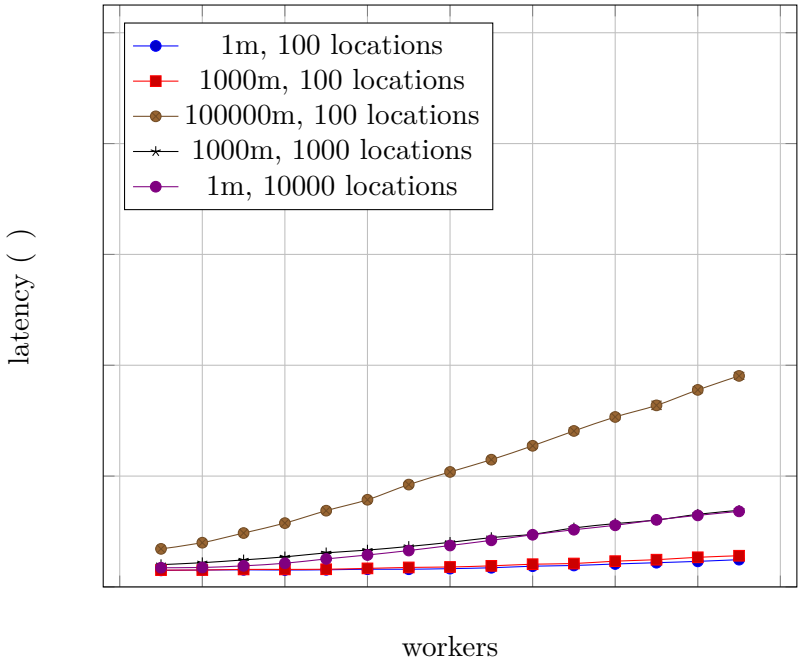


Figure 5.7: Influence of concurrency on latency (setup (a)).

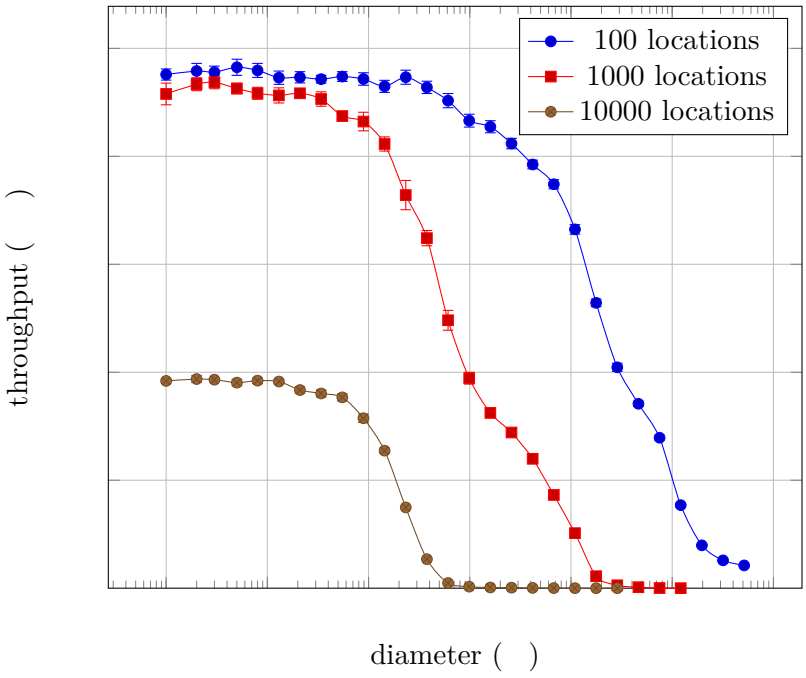


Figure 5.8: Throughput (setup (a)).

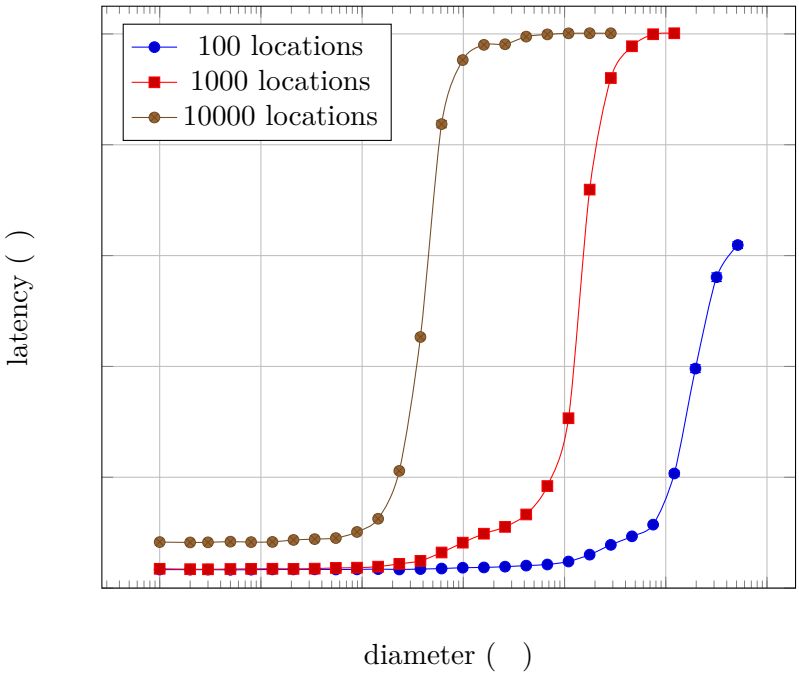


Figure 5.9: Latency (setup (a)).

and data set size have a significant influence on the results. Given that a throughput of around 240 queries per second is achieved for both the data set of 100 and 1000 locations with low diameter sizes, we can conclude that the performance is not limited by spatial computations at this point. Rather, it is likely that network performance or packet handling overhead are responsible for the bottleneck. Additionally, the graphs show that for large diameter sizes and large data set sizes, the throughput approaches 0 and the latency approaches 1 (the specified timeout value). This indicates that within the timeout limit, the system is not able to return a result with a large amount of matching locations. The performance for queries applied to the data set of 100 locations also appear to converge, but to different values. This can be explained by the fact that queries with diameter sizes larger than roughly 200 kilometers already encompass most of the location points in the data set, so increasing the diameter further will not increase the number of results.

The same experiment has been performed for nearest neighbor search as an alternative to querying specific diameters. The results of this for setup (a) are shown in Figure 5.10 and Figure 5.11. Similar to the results of the diameter tests, performance between the data set sizes of 100 and 1000 is close at lower nearest neighbor numbers. We also see that for the data sets with 1000 and 10000 locations, the system is again not able to supply an answer within the timeout limits, and throughput and latency converge to

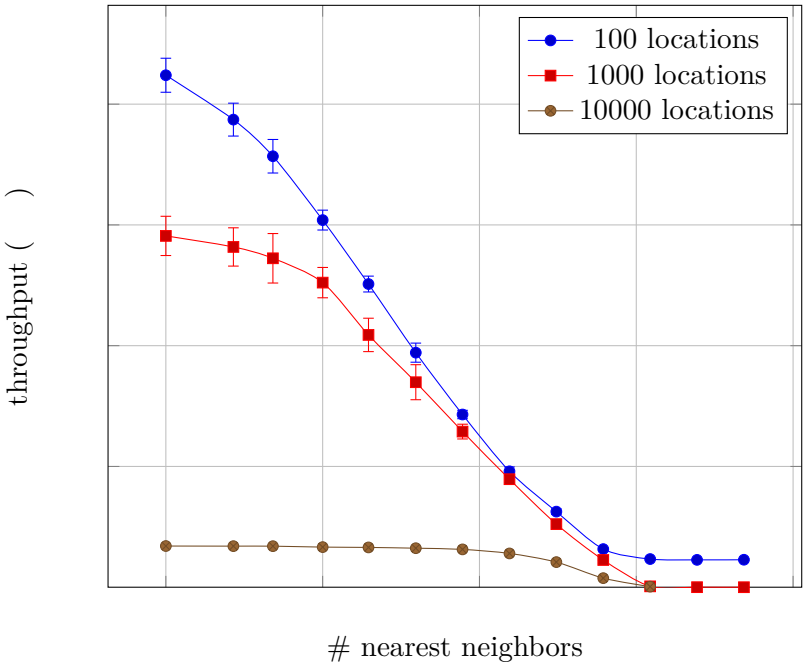


Figure 5.10: Throughput (nearest neighbors, setup (a)).

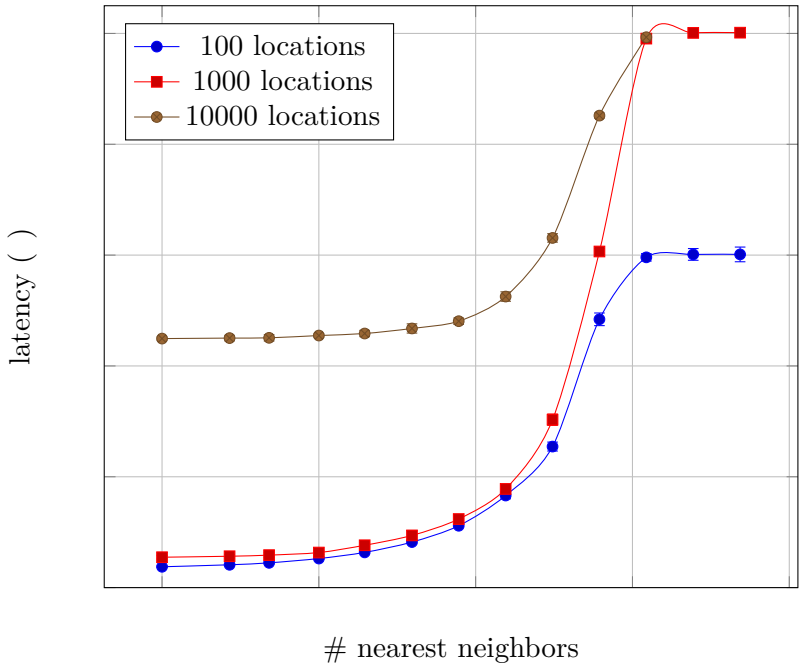


Figure 5.11: Latency (nearest neighbors, setup (a)).

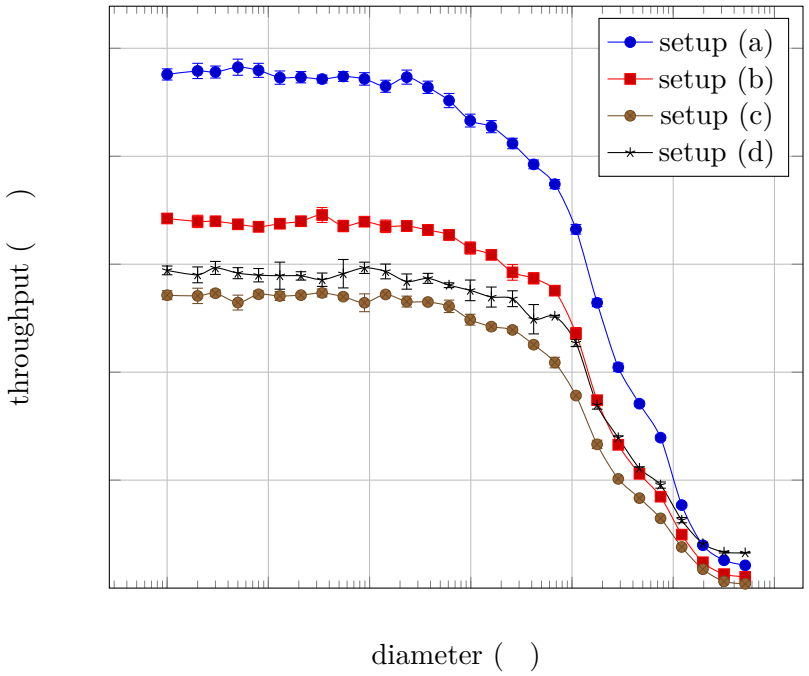


Figure 5.12: Throughput (100 locations).

their respective minimum and maximum values.

Equivalent comparisons have been made for other setups, but the graphs for these will be omitted because they show the same patterns, with larger data set sizes resulting in slower queries. Instead, we consider the performance of identical data set sizes between different setups. This is shown for the data set of 100 locations in Figure 5.12 and Figure 5.13. Performance for setups (b), (c) and (d) are almost always lower than that of setup (a), up to a point where they converge to a specific values. This result suggests that setup (b) and (c)'s larger number of servers that need to be contacted have a significant influence on performance for lower diameter values, while other factors become more dominant for larger diameters. The impact of larger diameters on setup (d) is lower, because it is able to distribute its computational load over two servers. Figure 5.14 and Figure 5.15 show a similar situation for 1000 locations.

With a data set of 10000 locations, shown in Figure 5.16 and Figure 5.17, the performance of setup (d) rises above that of the other setups for small diameters. This is likely the result of the system being able to divide its spatial computations over the two different edge nodes. At a diameter of around 100 meters, the performance becomes lower than setup (a)'s before converging to the same value.

The same comparison between setups for nearest neighbor queries is shown in Figure 5.18, Figure 5.19, Figure 5.20, Figure 5.21, Figure 5.22 and

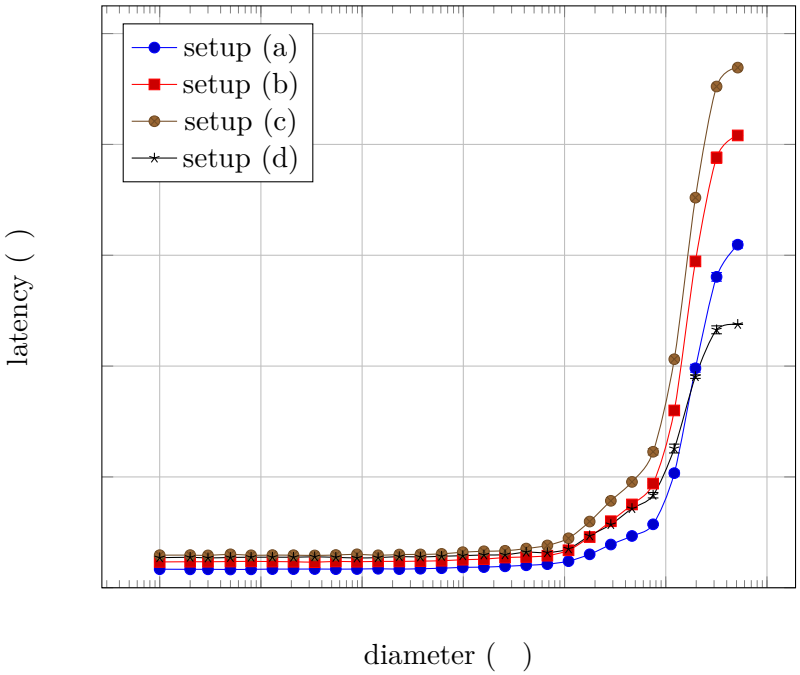


Figure 5.13: Latency (100 locations).

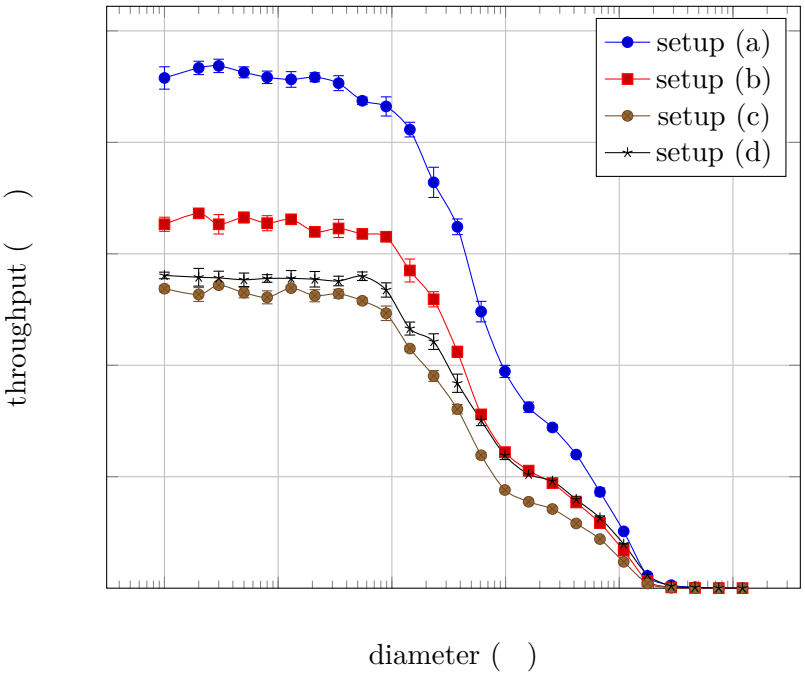


Figure 5.14: Throughput (1000 locations).

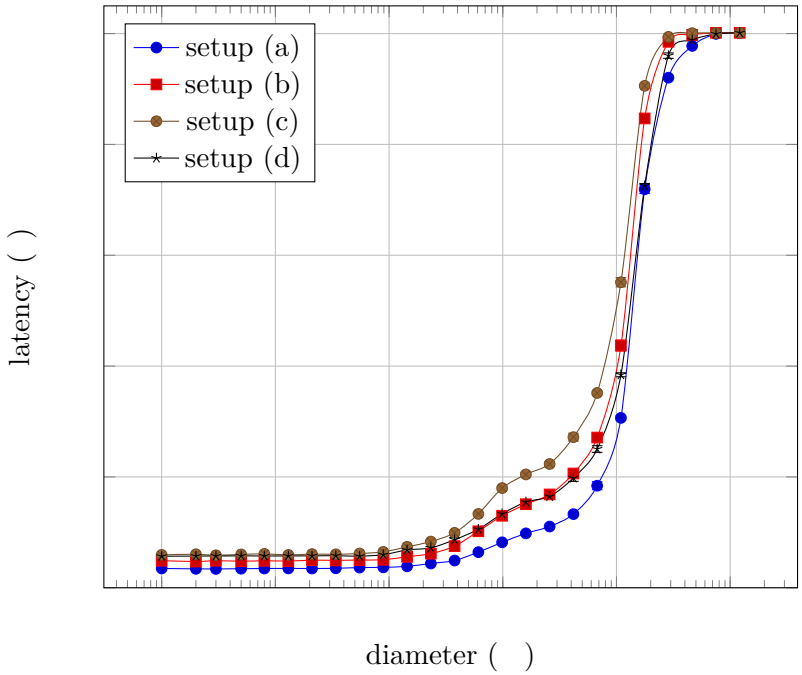


Figure 5.15: Latency (1000 locations).

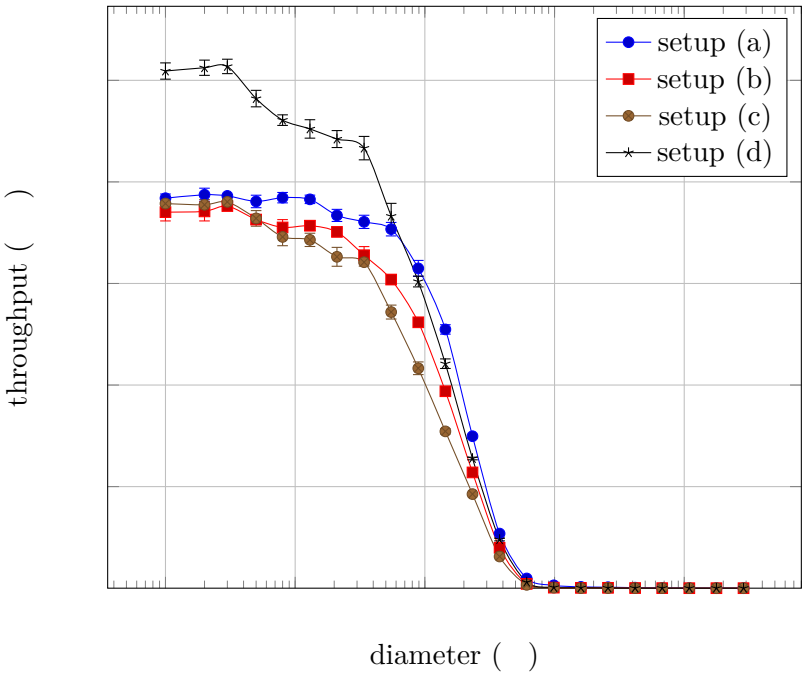


Figure 5.16: Throughput (10000 locations).

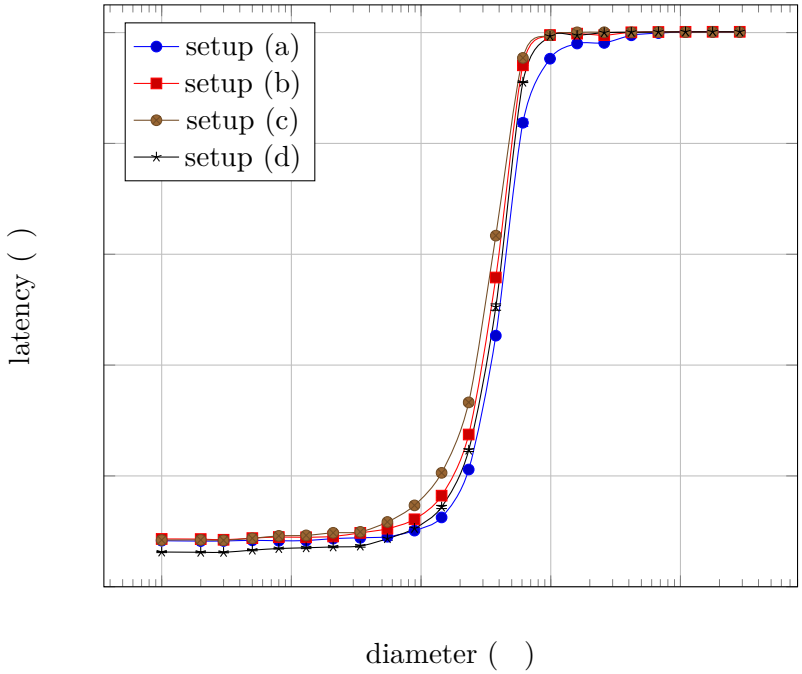


Figure 5.17: Latency (10000 locations).

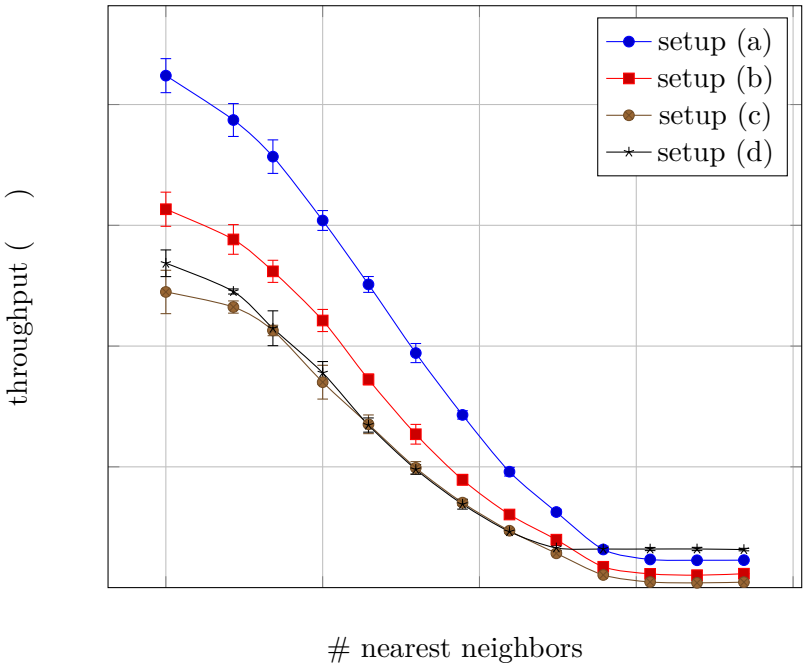


Figure 5.18: Throughput (nearest neighbors, 100 locations).

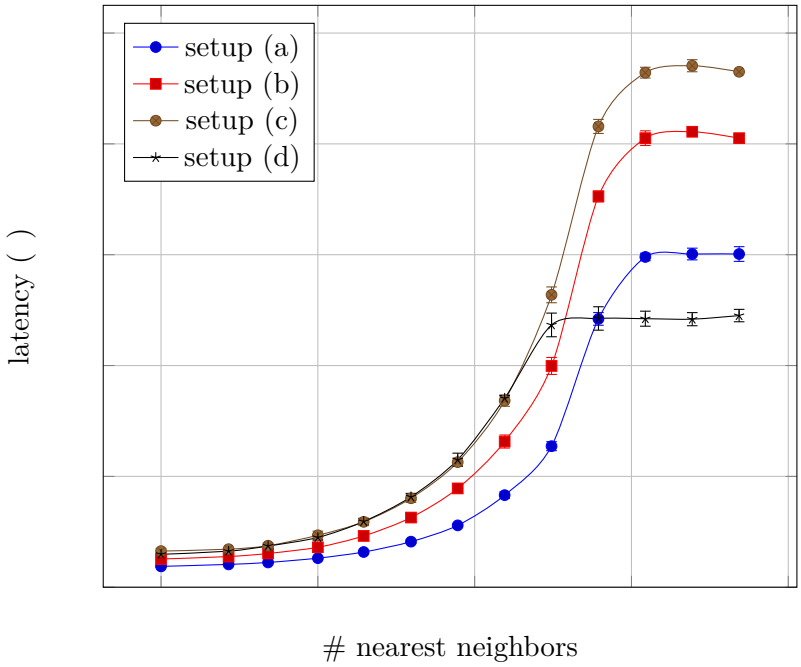


Figure 5.19: Latency (nearest neighbors, 100 locations).

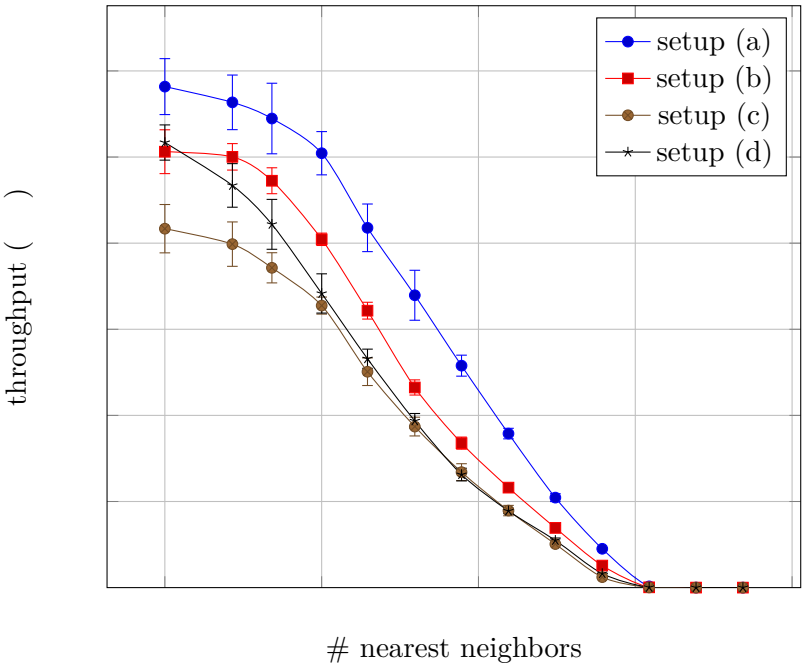


Figure 5.20: Throughput (nearest neighbors, 1000 locations).

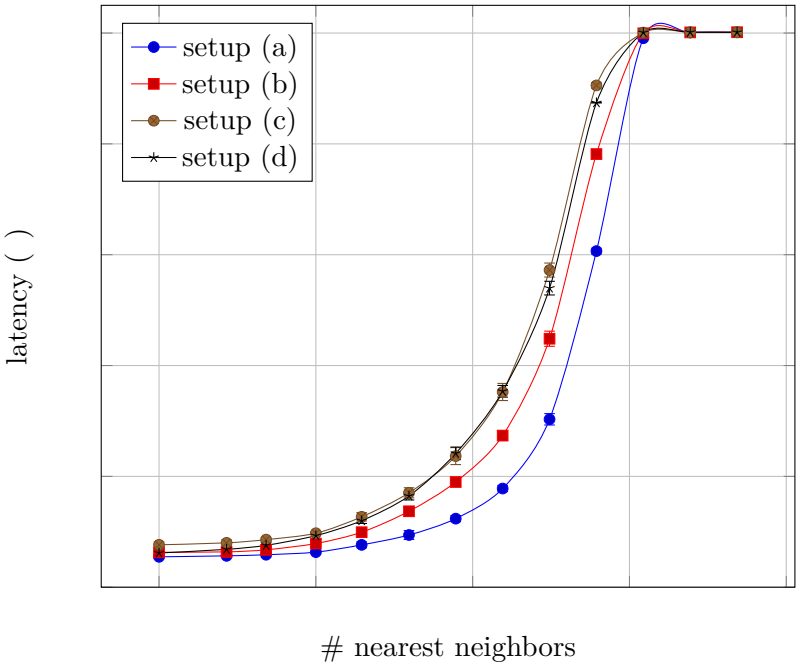


Figure 5.21: Latency (nearest neighbors, 1000 locations).

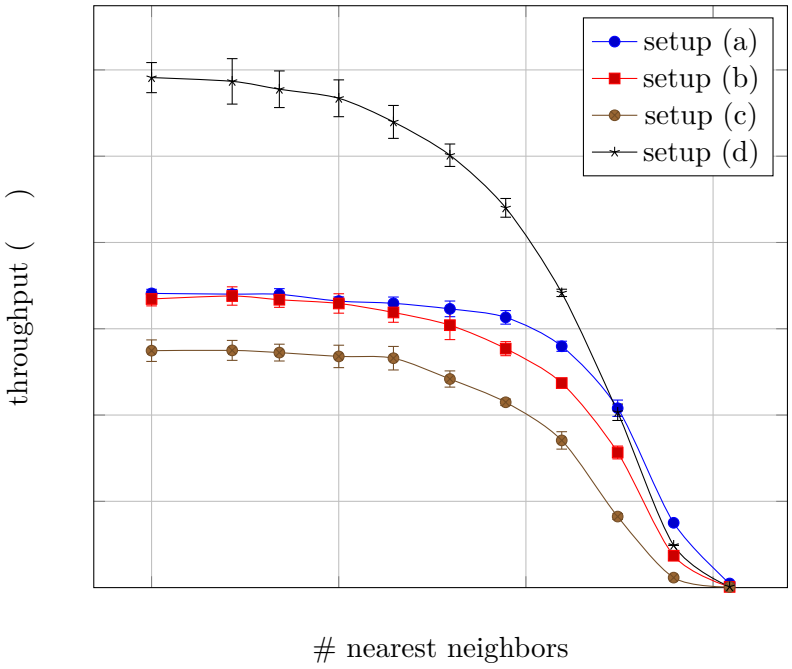


Figure 5.22: Throughput (nearest neighbors, 10000 locations).

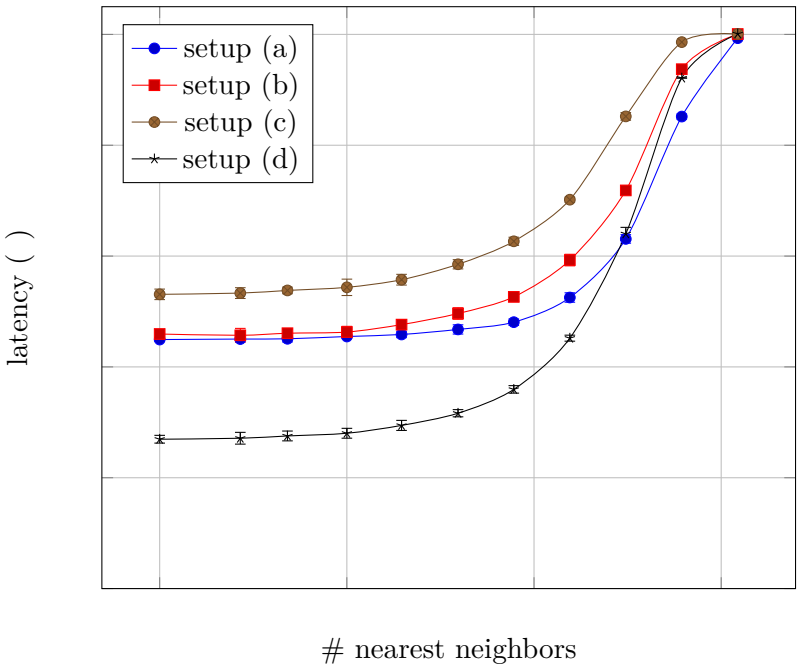


Figure 5.23: Latency (nearest neighbors, 10000 locations).

Figure 5.23. One interesting phenomenon occurs for the 100 data set size at around 50 requested nearest neighbors. At this point, the performance of setup (d) does not decrease further for higher requested numbers of nearest neighbors. The explanation for this is that setup (d) has its locations divided over two edge nodes. Because the parent server has no knowledge about which of its child servers contains the nearest neighbors, it asks both servers to give it their 50 nearest neighbors, out of their roughly 50 locations total each. Requesting any more nearest neighbors has no additional computational cost, because there simply are not more locations available. As such, the performance stays consistent.

Results for the other data set sizes roughly follow the patterns discovered in the evaluation for diameter-based queries. When a low number of LOC records is stored in the system, involving fewer servers in the resolution appears to be favored, because network delays have the largest impact on the performance. Queries performed on the system when it contains significantly more records tend to perform better on setup (d)'s load distribution, up to a point where the portion of packets that does not fit within the specified timeout window becomes large enough that setup (a)'s single server is able to more reliably provide a result.

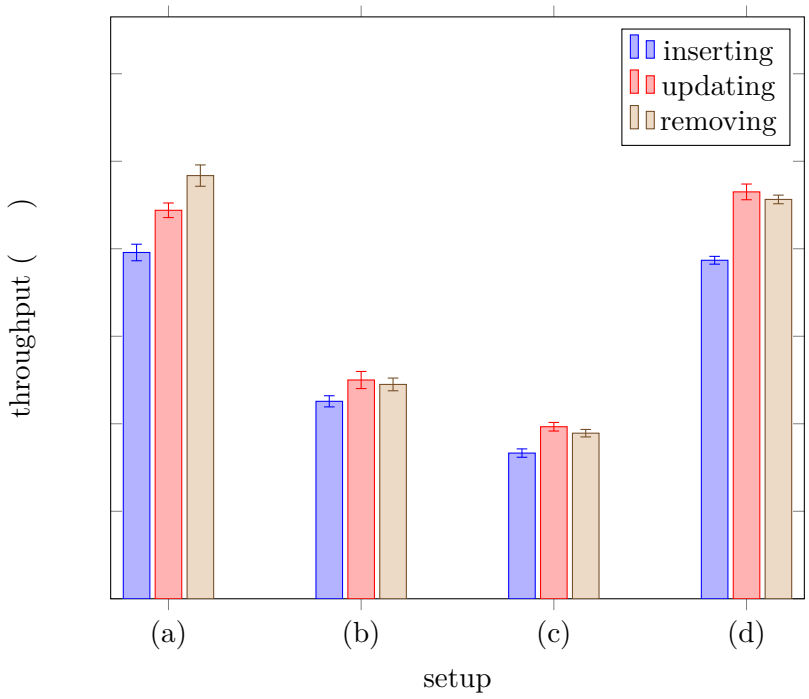


Figure 5.24: Updating throughput (100 locations).

5.3.2 Updating

We have evaluated three aspects of the dynamic location functionality: insertion, updating and removal. The insertion operation involves adding an A record with an IP address, as well as an entity's initial location in a LOC record. For updating a location, we assume that an entity's IP address has not changed. The updating operation therefore involves removing an old LOC record and replacing it with one containing a new location. The removal operation removes both the last know location in the form of a LOC record, but also the associated A record. The three different aspects therefore each involve two database operations. The aspects are evaluated as follows for a data set with x locations. For each location, a DNS Update message is sent that inserts both a fictional IP address and a location. Then, all locations are updated with individual DNS Update messages that remove the old LOC record and insert the new one. Finally, DNS Update messages are sent that remove all entities.

As with the query evaluation, all operations are executed with 8 concurrent workers to increase utilization. Storage of locations is also distributed over different edge nodes depending on geographical coordinates, as described in Section 5.2.

In Figure 5.24, Figure 5.25, Figure 5.26, Figure 5.27, Figure 5.28 and Figure 5.29, we show the performance of the three mentioned operations

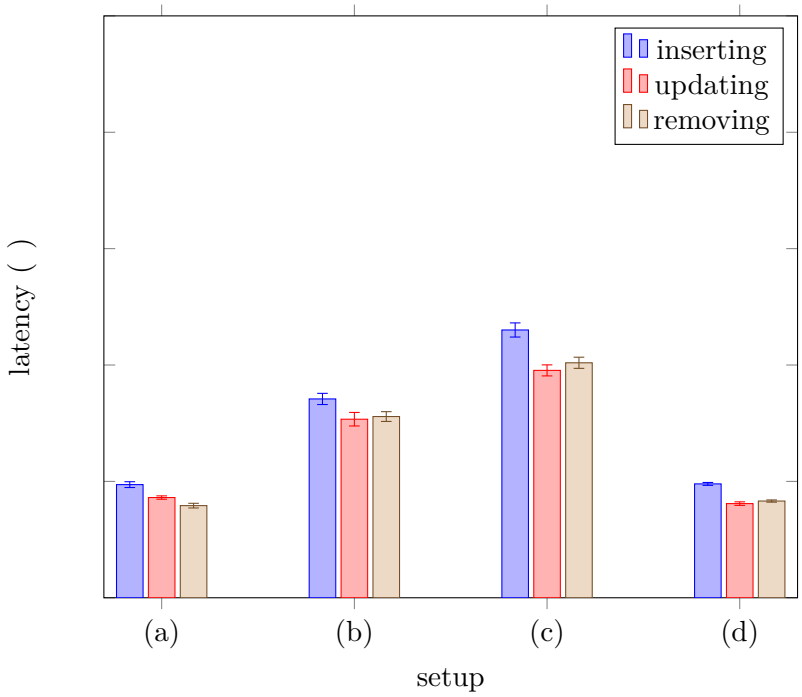


Figure 5.25: Updating latency (100 locations).

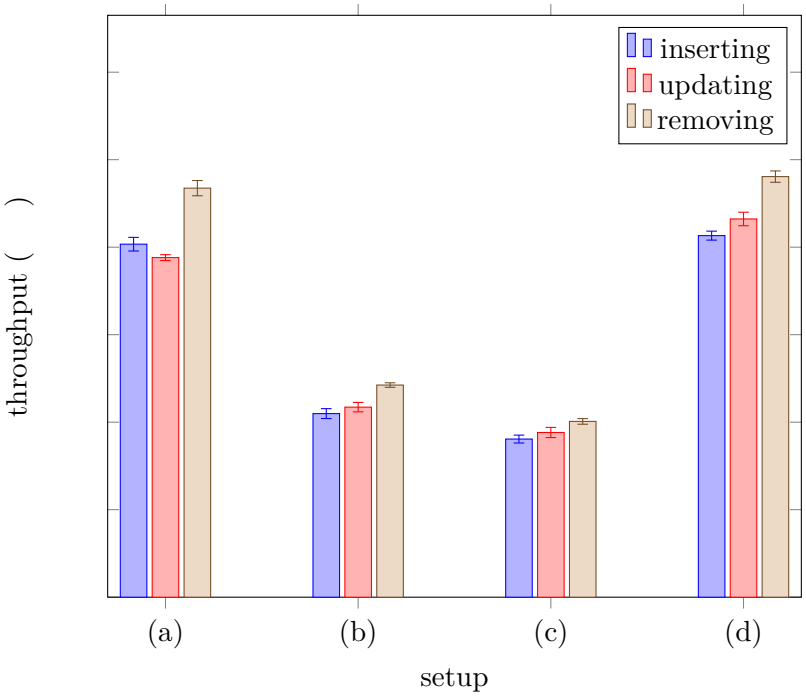


Figure 5.26: Updating throughput (1000 locations).

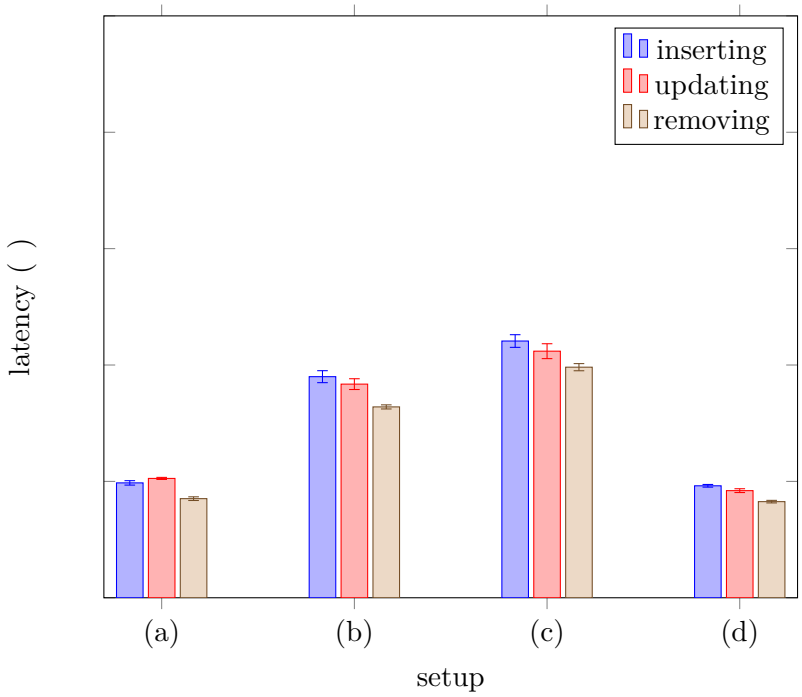


Figure 5.27: Updating latency (1000 locations).

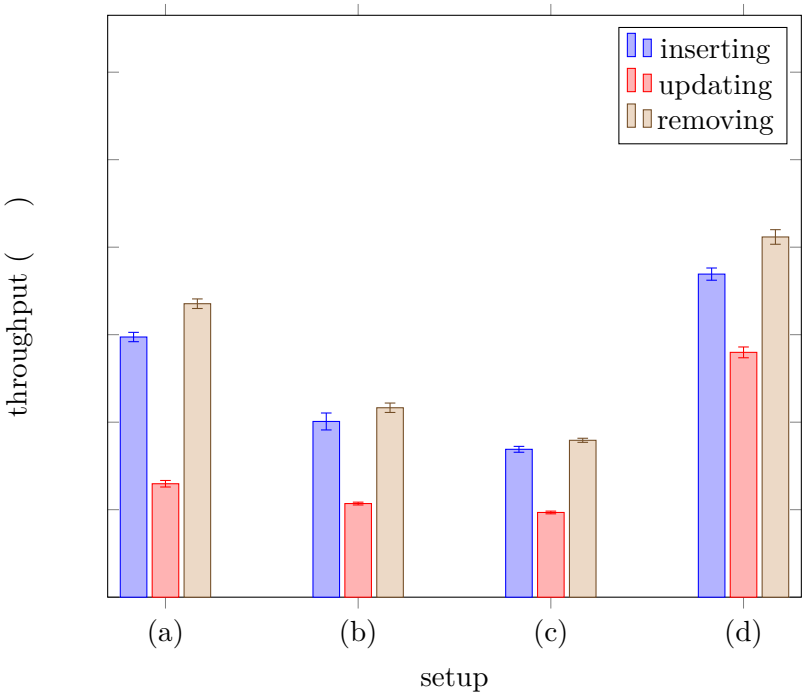


Figure 5.28: Updating throughput (10000 locations).

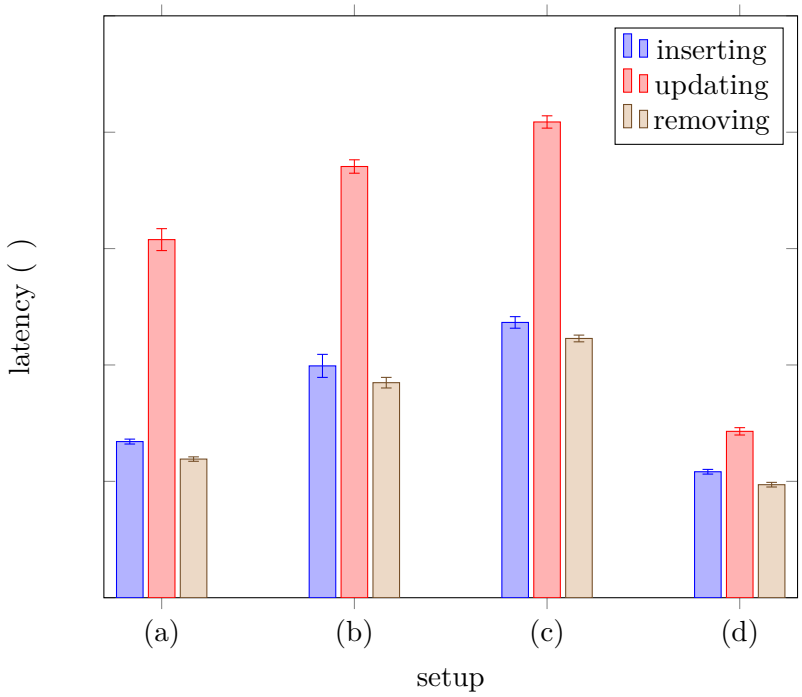


Figure 5.29: Updating latency (10000 locations).

when executed on different setups and using different data sets. We can see that on the data set with 100 locations, the insertion operation has the highest latency, followed by the updating operation. The removal operation is processed the fastest. As expected, introducing more depth to the server tree results in a higher latency, because each parent server needs to be updated with the renewed bounding boxes of its direct child servers. Setup (d) shows that the system scales well in width. Dividing LOC records over multiple servers appears to increase performance, even if it includes the extra operation of updating the parent server compared to setup (a). Updating locations becomes significantly more expensive when the amount of stored locations increases. Comparatively, the insertion and removal operations are not affected as much by a larger data set. At least part of this discrepancy can be explained by observing that the updating operation is always performed on a system that contains all locations in a data set of size . The insertion and removal operations work on a system that contains 0 to entities, depending on how many entities have already been inserted or removed.

5.4 Conclusions and Recommendations

We can make general conclusions about the performance of the system based on the results. For all evaluated combinations of input parameters, reduced overall performance was observed after introducing additional nodes into the setup by enlarging the vertical size of the domain name space tree. This is expected, as it increases the number of serves that need to be accessed, without providing any potential performance benefits. Conversely, because our evaluation method divides locations over all edge nodes, horizontally scaling the system with an additional server frequently improved performance. This allowed the system to balance its load over the multiple servers and utilize the additional resources. Having fewer servers involved still provides the best performance when a low number of locations is stored in the system, but at 10000 stored locations, multiple edge nodes typically provided the best throughput and latency. Based on these observations, we recommend that the introduction of additional servers should be considered when the number of stored locations per server exceeds 1000.

It is also possible for us to compare the location updating results to what would be required for our selected use case. Recall that in Section 5.2, we determined the maximum expected throughput to be 3.63 location updates per second, based on the busiest 60 second period in the data set. In total, the data set contains close to 1000 unique vehicles. Therefore, we look at the performance results of our evaluation with 1000 locations. This shows us that depending on the deployed setup, all operations can be performed with a throughput of 40 to 100 updates per second. This vastly exceeds the estimated use case requirement. Setup (a) and (d) show similar performance results for the chosen input parameters. However, as mentioned, setup (d)'s horizontal scaling would provide superior performance if Simacan were to consider tracking significantly more vehicles.

5.5 Towards large-scale adoption

The eDNS protocol as defined in previous works, as well as the additions made in our research, make significant changes to the standards defined in the DNS protocol. Although we have shown that eDNS support could be implemented in name server software using an abstraction layer, the query format and other elements of the protocol might be too big of a change to be considered for acceptance as a recognized DNS extension. Most of the issues are outside of the scope of our research, but in the following section we will briefly discuss some of the issues with the eDNS protocol as implemented in the prototype that would prevent it from being used in production, as well as possible solutions for these problems. To our knowledge, eDNS in its current state is not used in any production environments. Further changes

to the protocol that break backwards compatibility should therefore not have a significant impact.

Standard query format

The eDNS query format makes use of whitespace and parenthesis characters. This was purposely done to mimic the LOC record format, and to prevent non-eDNS queries from conflicting with eDNS queries. However, this choice might severely limit adoption. We could instead use a prefix, such as '`loc-`', using characters that are allowed in the DNS protocol to replace the parentheses. Likewise, spaces could be replaced with dashes ('-'). We provide this format as a suggestion for use in future eDNS research. In practice, a query could look as follows, representing the coordinate '51 55 37 N 5 12 52 E' with a diameter of 100 meters:

```
loc-51-55-37n5-12-52e100.domain
```

Alternatively, degrees with an arbitrary decimal precision could be used instead of the Degrees Minutes Seconds (DMS) format, with the dash replacing the decimal separator. For instance, referencing '51.927 N 5.214 E' with a diameter of 100 meters could be defined as:

```
loc-51-927n5-214e100.domain
```

Multiple label usage

Our research used multiple labels to represent complex shapes, based on a recommendation by previous work [48]. This usage conflicts with the hierarchical philosophy of DNS, where each label should represent a subsection of the label to its right. If we were to constrain our location specification to a single label, the allowed complexity of the shapes would be significantly lowered because of a label's size limitation (Section 2.3.3), but not restricted to only a circle. The following query could be used to used to create a line that follows the coordinates '51.927 N 5.214 E', '51.927 N 5.325 E', and '51.816 N 5.214 E', again with a diameter of 100 meters. We assume here that the standard query format change discussed previously has been applied as well.

```
loc-51-927n5-214e51-927n5-325e51-816n5-214e100.domain
```

If we were to reduce the precision of the coordinates, more than three coordinates could be listed within the size limitations of a label.

It may be possible to encode the coordinates in a way that is more efficient than the previously discussed format. One method involves storing the initial coordinate fully, but storing subsequent coordinates as an offset to either the first or the preceding coordinate. Given the fact that multiple coordinates in a query are likely in close proximity to each other, this would allow us to omit a coordinate's most significant figures.

Another approach would be to make use of the full range of characters allowed in DNS labels. As mentioned in Section 2.2.1, a label may make use of alphanumeric characters as well as the dash character. Although both upper case and lower case letters are allowed, the standard mentions that they should be treated as if identical [32]. Even with this limitation, the number of characters available for use exceeds the number of characters used to represent decimal numbers in text. This makes it theoretically possible to use a packing method for compression at the cost of human readability. Finding an optimal compression algorithm for this use case is outside of the scope of our work. Allowing the use of more characters would open the possibility for greater information density. A protocol to store binary labels in DNS exists [4], but is considered experimental due to difficulties in deploying support for it [5].

Distance pseudo-RR

Our nearest neighbor implementation makes use of a DST record based on the TXT record to communicate a result's distance to the query area between name servers. It is not stored in the zone file, and is therefore a pseudo-RR that is only attached to the 'additional data' section of a DNS request. A possibly more elegant way to implement this functionality would be to make use of the EDNS(0) OPT record [5]. This record already contains the pseudo-RR properties that are desired for the DST record. Only one OPT record is allowed per message, but it may include several options in the form of attribute-value pairs. Whether this record can fulfill the same role as the DST record can be researched in future work.

Privacy concerns

It may be considered undesirable, for privacy reasons, that the locations of every entity is accessible upon querying a server with a large enough area. These entities are uniquely identified by their hostname. For the purpose of sending messages to every entity within a geographical area, it might not be needed to get an identifiable hostname in addition to their corresponding IP addresses. We can consider obfuscating the hostnames to make them non-identifiable when queried, or even to return all results as having the query name as their names. For instance, if a client requests A records for `loc-51-927n5-214e100.domain`, all IP addresses would be returned as falling under the same `loc-51-927n5-214e100` hostname. Further research on the effects of this change should be done.

eDNS name

Although not directly related to compliance, we believe that changing the eDNS name to a more suitable one will aid in its acceptance. For one, the

eDNS name is easily confused with that of the EDNS(0) standard, a more widely used DNS extension. Secondly, the 'extended' part of the eDNS name does not cover the geographical aspect of the protocol well.

Chapter 6

Conclusions and Future work

In this report, we have described the design, development and evaluation of a functional eDNS system, preceded by extensive background research on the relevant technologies and concepts in Chapter 2. Recalling the research questions, listed in Section 1.1, we highlight here the conclusions of our research in order to fulfill them.

In previous implementations of eDNS, it was only possible to query areas with a circular or sperical shape. We have described an extension to the eDNS query format in Section 3.1, building on suggestions in previous works. This allows us to specify polygonal or line shapes as query areas. In practice, these shapes are better suited for addressing highways than circular shapes. The implementation details of this addition are discussed in Section 4.2.

To solve the problem of finding nearby results, we have described an approach (Section 3.2) and implemented the approach (Section 4.2.2) for nearest neighbor resolution functionality. This involves adding an argument to an eDNS query and introducing the DST pseudo-RR. Any number of nearby results can be requested. As explained in Section 3.2, this can be used in cases where vehicles need to be addressed via RSUs, but where RSUs provide incomplete coverage of an area.

We have reasoned about (Section 3.3, Section 3.5), and implemented (Chapter 4) eDNS in the PowerDNS DNS name server application, using a PostGIS enabled PostgreSQL database as a back-end. Using a database back-end with geographical capabilities prevents an eDNS name server application from needing to implement complex spatial operations. In addition, several DNS name server applications already use a database for resource record storage. Implementation of eDNS can therefore be simplified.

After previous works have shown that updating individual entities with new locations is possible, we have described in Section 3.4, and subsequently implemented the concept of dynamicity in eDNS using the standardized DynDNS method (Section 4.2.1). This allows us to track the locations of vehicles in terms of LOC resource records. It involves keeping the BND

resource record updated, and notifying parent name servers about updated coverage areas.

The performance of various eDNS operations is measured in terms of the throughput and latency metrics, as described in Section 5.2.1. Besides evaluating both the querying and updating functionality of eDNS, we evaluate different server setups and input parameters (Section 5.2.2). In particular, the influence of the size of the location data set, as well as the diameter of queried areas, is tested extensively. Furthermore, the number of concurrent requests has been shown in Section 5.3 to influence the performance.

Performance has been evaluated for queries and updates in Section 5.3.1 and Section 5.3.2. The performance is shown to be strongly dependent on server setup, as well as the input parameters described in Section 5.2.2 — this makes it difficult to summarize the performance evaluation of the system. However, the test results show that in most of the evaluated scenarios, scaling the width of server setups frequently influences the performance positively, while scaling in height always influences the performance negatively. Distributing LOC records over multiple servers allows the system to perform its calculations faster, potentially improving both throughput and latency.

Section 3.6 addresses two possible use cases for eDNS in Simacan products. The first use case involves storing the locations of RSUs in an eDNS system, while the second stores locations of individual OBUs in vehicles. We have focused on the second use case in our research, because it allows us to evaluate the dynamicity aspect of our work.

We can compare the location updating requirements of our selected use, described in Section 5.2, to the results of our evaluation in Section 5.3.2. The number of vehicles that would need to be tracked is close to 1000, which corresponds to one of our tested location data set sizes. Depending on the chosen server setup, the evaluation indicates that our eDNS system supports 40 to 100 updates per second. This vastly exceeds the expected maximum update rate of 3.63 updates per second.

We have discussed several potential modifications to the eDNS protocol in Section 5.5. Notably, we discuss changing the query format to increase compliance with the existing DNS protocol. However, while we have made several suggestions to help the adoption of eDNS, we have not evaluated them extensively.

6.1 Future work

As mentioned, we have briefly touched on potential modifications of the eDNS protocol to increase the compatibility with the DNS protocol in Section 5.5. This includes changing the query format to only include characters that are allowed by default; restricting the geographical part of a query to a single DNS label; implementing the communication of distances through the

existing OPT record; considering privacy; and changing the protocol name. The viability of the proposed changes can be evaluated in future work. We believe that this is an important step to increase eDNS acceptance.

The scalability of the system can be evaluated more extensively. In our evaluation the total number of employed servers has been restricted to 3 for budget reasons. Due to the need for a single apex node in a tree, the maximum number of servers in one tree level was 2. Evaluating horizontal scalability further could provide new insights.

Additionally, future work may focus on the performance of the protocol and implementation. For one, performance of our prototype and the older NSD based implementation can be compared. Secondly, although we have shown our prototype to exceed the requirements of our chosen use case, possible performance improvements may be researched to enable more complex use cases. In our work, we have considered geocasting to RSUs and OBUs. Keeping track of the location of every single vehicle on the road networks on a global scale will greatly increase the performance requirements. In the context of IoT, it may also be of significant interest to be able to store the locations of other entity types.

Bibliography

[1] R. Baldessari, B. Bödekker, M. Deegener, A. Festag, W. Franz, C. C. Kellum, T. Kosch, A. Kovacs, M. Lenardi, C. Menig, et al. Car 2 car communication consortium manifesto. Technical report, CAR 2 CAR Communication Consortium, 2007. `https://www.car-2-car.org/index.php?id=31`.

[2] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. *The R*-tree: an efficient and robust access method for points and rectangles*, volume 19. ACM, 1990.

[3] A. Buchenscheit, F. Schaub, F. Kargl, and M. Weber. A vanet-based emergency vehicle warning system. In *Vehicular Networking Conference (VNC), 2009 IEEE*, pages 1–8. IEEE, 2009.

[4] M. Crawford. Binary labels in the domain name system. RFC 2673, RFC Editor, August 1999. `http://www.rfc-editor.org/rfc/rfc2673.txt`.

[5] J. Damas, M. Graff, and P. Vixie. Extension mechanisms for dns (edns(0)). STD 75, RFC Editor, April 2013. `http://www.rfc-editor.org/rfc/rfc6891.txt`.

[6] C. Davis, P. Vixie, T. Goodwin, and I. Dickinson. A means for expressing location information in the domain name system. RFC 1876, RFC Editor, January 1996. `http://www.rfc-editor.org/rfc/rfc1876.txt`.

[7] A. Eldawy and M. F. Mokbel. The era of big spatial data. In *Proceedings of the International Workshop of Cloud Data Management CloudDM 2015 co-located with ICDE 2015*, 2014.

[8] T. Ernst. Final geonet architecture design, January 2010. `http://www.geonetproject.eu/?download=GeoNet-D1.2-architecture_design.pdf`.

[9] ETSI. Intelligent transport systems (its); european profile standard for the physical and medium access control layer of intelligent transport systems operating in the 5 ghz frequency band, November 2009.

[10] T. Fioreze and G. Heijenk. Extending dns to support geocasting towards vanets: A proposal. In *Vehicular Networking Conference (VNC), 2010 IEEE*, pages 271–277. IEEE, 2010.

[11] T. Fioreze and G. Heijenk. Extending the domain name system (dns) to provide geographical addressing towards vehicular ad-hoc networks (vanets). In *Vehicular Networking Conference (VNC), 2011 IEEE*, pages 70–77. IEEE, 2011.

[12] I. T. Force. Observations on geonetworking, November 2012. `http://ec.europa.eu/information_society/newsroom/cf/dae/document.cfm?doc_id=1934`.

[13] O. Gudmundsson and M. Majkowski. Deprecating the dns any meta-query type. `https://blog.cloudflare.com/deprecating-dns-any-meta-query-type/`, March 2015.

[14] A. Guttman. *R-trees: a dynamic index structure for spatial searching*, volume 14. ACM, 1984.

[15] T. Hain. An ipv6 geographic global unicast address format. Internet-Draft draft-hain-ipv6-geo-addr-02, IETF Secretariat, July 2010. `https://tools.ietf.org/id/draft-hain-ipv6-geo-addr-02.txt`.

[16] J. R. Herring. Opengis implementation specification for geographic information - simple feature access - part 2: Sql option. *Open Geospatial Consortium Inc*, 2006.

[17] J. R. Herring. Opengis implementation specification for geographic information - simple feature access - part 1: Common architecture. *Open Geospatial Consortium Inc*, 2011.

[18] G. R. Hjaltason and H. Samet. Distance browsing in spatial databases. *ACM Transactions on Database Systems (TODS)*, 24(2):265–318, 1999.

[19] B. Hubert. New documentation on powerdns backends, and what they are and aren't. `http://mailman.powerdns.com/pipermail/pdns-dev/2015-June/001509.html`, June 2015.

[20] ICANN. Reconsideration request 00-14. `https://archive.icann.org/en/committees/reconsideration/rc00-14.htm`, March 2001.

[21] T. Imieliński and J. C. Navas. Gps-based addressing and routing. RFC 2009, RFC Editor, November 1996. `http://www.rfc-editor.org/rfc/rfc2009.txt`.

[22] T. Imieliński and J. C. Navas. Gps-based geographic addressing, routing, and resource discovery. *Communications of the ACM*, 42(4):86–92, 1999.

[23] A. Intl. Standard specification for telecommunications and information exchange between roadside and vehicle systems-5 ghz band dedicated short range communications (dsrc). *Medium Access Control and Physical Layer specifications, E2213-03*, 2003.

[24] C. S. Jensen, J. Kolářvr, T. B. Pedersen, and I. Timko. Nearest neighbor queries in road networks. In *Proceedings of the 11th ACM international symposium on Advances in geographic information systems*, pages 1–8. ACM, 2003.

[25] T. Jochle, B. Wiedersheim, F. Schaub, and M. Weber. Efficiency analysis of geocast target region specifications for vanet applications. In *VNC*, pages 250–257, 2012.

[26] G. Karagiannis, O. Altintas, E. Ekici, G. Heijenk, B. Jarupan, K. Lin, and T. Weil. Vehicular networking: A survey and tutorial on requirements, architectures, challenges, standards and solutions. *Communications Surveys & Tutorials, IEEE*, 13(4):584–616, 2011.

[27] A. Kovacs. Final geonet specification, January 2010. `http://www.geonetproject.eu/?download=GeoNet-D2.2-final_specification.pdf`.

[28] C. Liu and P. Albitz. *DNS and Bind*. O'Reilly Media, Inc., 2006.

[29] C. Maihofer. A survey of geocast routing protocols. *Communications Surveys & Tutorials, IEEE*, 6(2):32–42, 2004.

[30] J.-P. Mens. *Alternative DNS Servers*. UIT Cambridge Ltd., 2009.

[31] P. Mockapetris. Domain names - concepts and facilities. STD 13, RFC Editor, November 1987. `http://www.rfc-editor.org/rfc/rfc1034.txt`.

[32] P. Mockapetris. Domain names - implementation and specification. STD 13, RFC Editor, November 1987. `http://www.rfc-editor.org/rfc/rfc1035.txt`.

[33] M. Ohta. Incremental zone transfer in dns. RFC 1995, RFC Editor, August 1996. `http://www.rfc-editor.org/rfc/rfc1995.txt`.

[34] B. Paul, M. Ibrahim, M. Bikas, and A. Naser. Vanet routing protocols: Pros and cons. *arXiv preprint arXiv:1204.1201*, 2012.

[35] J. Postel. Domain name system structure and delegation. RFC 1591, RFC Editor, March 1994. `http://www.rfc-editor.org/rfc/rfc1591.txt`.

[36] P. Ramsey. Indexed nearest neighbour search in postgis. http://boundlessgeo.com/2011/09/indexed-nearest-neighbour-search-in-postgis/, September 2011.

[37] W. Rodger. Schism hits domain name system. http://connection.ebscohost.com/c/articles/9707181217/schism-hits-domain-name-system, February 1997.

[38] R. Rosenbaum. Using the domain name system to store arbitrary string attributes. RFC 1464, RFC Editor, May 1993. http://www.rfc-editor.org/rfc/rfc1464.txt.

[39] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *ACM sigmod record*, volume 24, pages 71–79. ACM, 1995.

[40] E. Schoch, F. Kargl, M. Weber, and T. Leinmuller. Communication patterns in vanets. *Communications Magazine, IEEE*, 46(11):119–125, 2008.

[41] S. Steiniger and A. J. Hunter. The 2012 free and open source gis software map–a guide to facilitate research, development, and adoption. *Computers, Environment and Urban Systems*, 39:136–150, 2013.

[42] J. van Leeuwen. Dynamicity management in domain name system resource records. Bachelorreferaat, University of Twente, 2014.

[43] R. Vaughn and G. Evron. Dns amplification attacks. Technical report, March 2006.

[44] P. Vixie. A mechanism for prompt notification of zone changes (dns notify). RFC 1996, RFC Editor, August 1996. http://www.rfc-editor.org/rfc/rfc1996.txt.

[45] P. Vixie. Extension mechanisms for dns (edns0). RFC 2671, RFC Editor, August 1999. http://www.rfc-editor.org/rfc/rfc2671.txt.

[46] P. Vixie, O. Gudmundsson, D. E. Eastlake 3rd, and B. Wellington. Secret key transaction authentication for dns (tsig). RFC 2845, RFC Editor, May 2000. http://www.rfc-editor.org/rfc/rfc2845.txt.

[47] P. Vixie, S. Thomson, R. Yakov, and J. Bound. Dynamic updates in the domain name system (dns update). RFC 2136, RFC Editor, April 1997. http://www.rfc-editor.org/rfc/rfc2136.txt.

[48] M. Westra. Extending the domain name system with geographically scoped queries. Master's thesis, University of Twente, 2013.