OpenFLAME: A Federated Spatial Naming Infrastructure

Sagar Bharadwaj
Carnegie Mellon University, Pittsburgh, USA

Ziyong Ma
Carnegie Mellon University, Pittsburgh, USA

Ivan Liang
Carnegie Mellon University, Pittsburgh, USA

Michael Farb
Carnegie Mellon University, Pittsburgh, USA

Anthony Rowe
Carnegie Mellon University, Pittsburgh, USA

Srinivasan Seshan
Carnegie Mellon University, Pittsburgh, USA

Abstract

Spatial applications, i.e., applications that tie digital information with the physical world, have improved many of our daily activities, such as navigation and ride-sharing. This class of applications also holds significant promise of enabling new industries such as augmented reality and robotics. The development of these applications is enabled by a system that can resolve real-world locations to names, or a spatial naming system. Today, mapping platforms provided by organizations like Google and Apple serve as spatial naming systems. These maps are centralized and primarily cover outdoor spaces. We envision that future spatial applications, such as persistent world-scale augmented reality, would require detailed and precise spatial data across indoor and outdoor spaces. The scale of cartography efforts required to survey indoor spaces and their privacy needs inhibit existing centralized maps from incorporating such spaces into their platform.

In this paper, we present the design and implementation of *OpenFLAME*[1], a federated spatial naming system, or in other words, a federated mapping infrastructure. *OpenFLAME* enables independent parties to manage and serve their own maps of physical regions. This unlocks scalability of map management, isolation, and privacy of maps. The discovery system that identifies maps hosted at a given location is a primary component of *OpenFLAME*. We implement *OpenFLAME* on top of the existing Domain Name System (DNS), which enables us to leverage its existing infrastructure. We implement map services such as address-to-location mapping, routing, and localization on top of our federated mapping infrastructure.

2012 ACM Subject Classification Computer systems organization → Distributed architectures; Networks → Naming and addressing

Keywords and phrases Geographic naming system, Augmented reality, Robotics

Digital Object Identifier 10.4230/OASIcs.NINeS.2026.20

Acknowledgements This material is based upon work supported by the National Science Foundation under Grant No. CNS-1956095. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

[1] OpenFLAME stands for Open Federated Localization and Mapping Engine

1st New Ideas in Networked Systems (NINeS 2026).
Editors: Katerina J. Argyraki and Aurojit Panda; Article No. 20; pp. 20:1–20:26
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Spatial applications, i.e. applications that utilize spatial data and tie digital information to the physical world, have revolutionized many industries such as navigation, ride-sharing, product delivery, and transportation. They hold the promise of enabling emerging industries such as world-scale augmented reality and autonomous robots. While the vision of spatial applications is promising, the reality is that such applications are hard to build today because they lack an underlying infrastructure to discover and reference spatial content.

A well-designed naming infrastructure—a system for identifying and discovering entities in the system—would enable spatial applications to easily reference relevant content. The DNS, for example, was a key enabler of the Web and the Internet at large. It provided a simple mechanism for converting human-readable names (domain names) to server IP addresses. Similarly, we believe spatial applications need a spatial naming system to relate human-readable names (e.g., Louvre Museum) with real-world locations (e.g., 48°N and 2.3°E) and the content associated with those locations (e.g., museum collections). Since this naming system translates names to physical locations, we use the term *map* to refer to it.

The structure of a naming system plays a critical role in shaping and limiting the functionality of any distributed system built upon it. For example, the way a naming system handles the addition of new entities can introduce bottlenecks that hinder scalability and maintenance. The expansion of the Web and the broader Internet in their early days was partly due to the federated and pseudo-decentralized nature of the DNS, which enabled organizations to independently manage their participation on the network. This relationship between naming systems and application constraints also applies to spatial applications. For example, the centralized and single-owner nature of today's spatial naming systems or digital maps, such as Google and Apple maps, limits their functionality.

An example of the limitations that centralized maps face is that only information gathered and exposed by organizations maintaining them is available to applications. Extending spatial applications indoors is a use case that highlights the importance of a federated mapping infrastructure. Indoor maps contain sensitive information that needs to be owned and controlled by the owner of the physical space. For example, many organizations would benefit from providing accurate indoor map of their private offices and integrating it with outdoor maps for applications such as office navigation for their employees, but may not be willing to publicly host detailed maps. In addition, the effort required for the storage and cartography of indoor spaces far exceeds that of outdoor maps [7] and surveying this space will likely be impractical for any single centralized organization. While crowd-sourcing into centralized maps is a possible solution, many organizations would not cede control of their maps to centralized organizations.

In this paper, we describe the design and implementation of *OpenFLAME*, a federated mapping system. The fundamental units of *OpenFLAME* are *map servers*—independent systems deployed by potentially disparate parties that provide map data and services confined to a physical region. Some map servers may correspond to existing large-scale providers such as Google Maps, while others may be independently operated to serve indoor or private regions controlled by individual organizations. Importantly, our system does not attempt to replace large-scale centralized map providers but rather aims to allow smaller independent map providers to interoperate with them, encouraging larger coverage. *OpenFLAME* provides the means to efficiently discover map servers relevant for a region and combine information from these servers to support services like name-to-location translation, routing, and localization that are essential to spatial applications. Our contributions are:

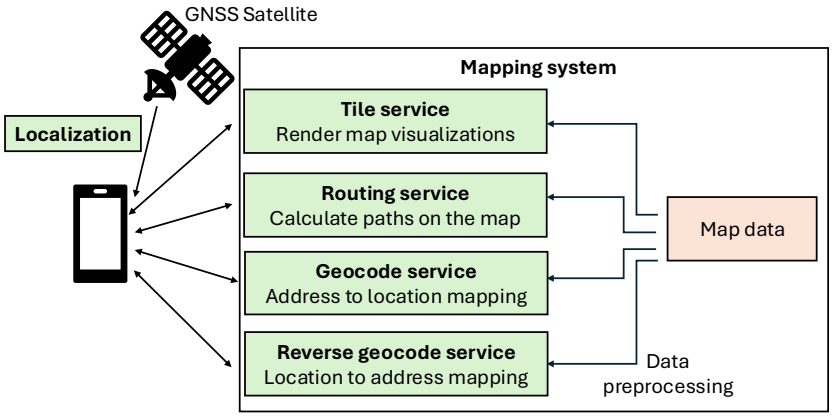


Figure 1 Centralized map architecture.

- We present the first design of a federated mapping infrastructure. Our infrastructure can support heterogeneous maps, has a low barrier to entry, ensures map privacy, and enables fine-grained access control (§ 3).
- We present a DNS-based implementation of the federated mapping system (§ 4). The advantage of leveraging the DNS is that we have access to its existing deployment and caching mechanisms. We describe how we convert location-based queries to DNS lookups. We also show how arbitrary regions representing map boundaries can be registered on the existing DNS as name records.
- We implement location-based services on top of our mapping infrastructure (§ 5). Specifically, we describe our implementations for address-to-location mapping (§ 5.2), routing (§ 5.3), and localization (§ 5.4).

We evaluate the discovery system and map services and compare them to centralized services in § 7.[2]

2 Example application

To better understand the needs of future spatial applications, we start by describing what we consider a typical such application, campus navigation. While we use this application to describe and motivate our design, our goal is to support all future spatial applications.

Let us consider a scenario where a user wishes to get pedestrian navigation guidance from their location in a typical city neighborhood to a specific professor's office at a nearby university campus. Today, such an application would rely on a centralized mapping platform such as Google Maps to search and locate the destination. It would also use a combination of technologies to determine the location of a user, including GPS, image data from Google Street View, and WiFi/Cellular signal strength. Figure 1 shows the architecture of a centralized mapping infrastructure. It includes a centrally maintained map database which is preprocessed into different forms to enable location-based services. For example, it is processed into a graph so the routing service can run shortest path algorithms on it. Each service uses the preprocessed data to serve application requests. Such a centralized map would likely only have public landmarks and outdoor walkways. The professor's office, or the

[2] We demo some tools built for *OpenFLAME* on `https://www.open-flame.com/`.

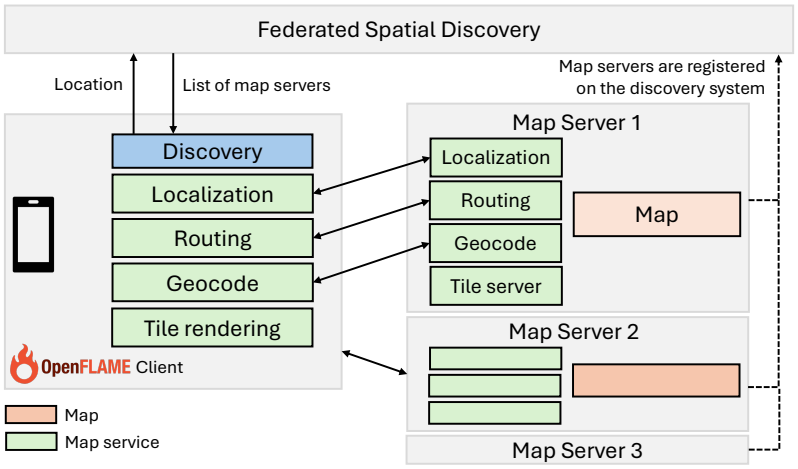


Figure 2 *OpenFLAME* architecture.

university hallways that lead to the office might not be part of the map unless the centralized service had surveyed the university.

Ideally, we would like the application to provide precise visual guidance along all steps of the path. Unfortunately, existing applications fail to meet this objective in multiple ways – failing to provide precise guidance when localization is inaccurate, or in this case, failing to provide complete guidance as the required destination is not in the map database.

3 Design

Figure 2 shows the simplified architecture of *OpenFLAME*. Maps of different regions are stored on separate *map servers* maintained by independent organizations. Map servers also provide location-based services on top of the maps that they store. An *OpenFLAME* client discovers potentially zero or more map servers covering the region of interest using a federated spatial database. It then contacts these discovered servers to obtain services such as routing and localization, combining results from multiple maps when needed. Alternative design choices where the discovery system discovers spatial content or applications associated with a location are discussed in Appendix A. In this section, we discuss our design decisions pertaining to the organization of map data (§ 3.1), discovery query model (§ 3.2), and security and privacy models (§ 3.3). We will describe the implementation of these models in the next section (§ 4).

3.1 Data Model

3.1.1 Abstractions: Maps and Map Servers

Map server. A map server is a system that stores the *map* of a region and provides *map services* on it. For example, a university building's map server would store all the offices and navigable hallways in the building and might provide an image-based localization service to support AR applications. A map server can impose fine-grained security and privacy policies on users and applications.

Map. A Map is a representation encoding relationships and attributes of spatial entities in a geographic region. While traditionally, a map refers to the visual representation of

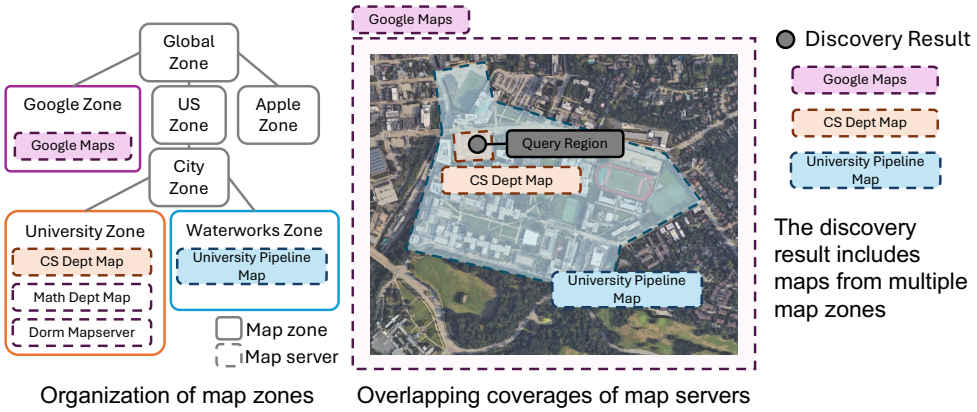


Figure 3 *OpenFLAME* data organization and query model.

geographic features, in our context it is the data that underlies such visual representations. We do not restrict the format in which map data is stored in individual map servers.

Map services. Location-based services built on top of maps are called map services. The green boxes in Figure 2 show some examples of map services. The tile service, for example, returns a visual representation of the map. Clients access map data only through map services.

Map zone. A set of map servers are grouped to form a map zone for organizational convenience and ease of delegation. For example, the map servers of the different departments of a university form the map zone for the university.

Each map server is registered under a zone. Both map zones and servers define their own coverage—the spatial extent they are responsible for. The coverage of a map server must lie entirely within the coverage of its zone. However, the coverage of a zone may extend beyond the combined coverage of its servers, meaning that zones can include "empty" regions that are not served by any map server. A zone can delegate the responsibility of parts of its coverage down to sub-zones. A zone can be registered with one or more parent zones.

For example, consider a university setting up its map on *OpenFLAME*. The university first defines the coverage of its zone, which spans all buildings on its campus. This allows the university to manage its zone independently of other map zones. It registers its zone with a parent zone, e.g., the city zone. Initially, much of the university zone's coverage may consist of empty regions not served by any map server. Over time, individual university departments can populate the zone by registering their own servers, each covering their respective areas. Within the zone, these map servers operate independently of one another and their coverages can overlap with each other. The university facilities department, for example, could set up its own sub-zone and maintain all of its maps (e.g., electricity and plumbing plans) within this sub-zone.

The fundamental unit of discovery for spatial applications is the map server. Map zones exist primarily for organizational and administrative convenience of delegation. Applications do not need to interact with zones directly. However, zones can serve as an optional filtering mechanism; i.e., applications may restrict discovery to specific zones if they wish.

3.1.2 Organization of Map Servers and Zones

Map zones form a nested inclusion hierarchy; a child zone must be completely contained within its parent zone. The coverage of map zones can overlap with each other. Map servers

are the leaves of this hierarchy. They always have a parent zone, and their coverages may also overlap.

An obvious starting point for organizing the hierarchy of map zones would be to reuse existing geopolitical hierarchies such as countries, states, and cities [21]. However, such traditional hierarchies are not suited for our purpose. First, they are fraught with disputes. National boundaries are contested, and even property lines are frequent sources of legal conflict. Anchoring a technical system to these boundaries risks inheriting political disagreements [11]. Second, geopolitical hierarchies generally assume exclusive ownership, where a region belongs to exactly one parent. This prevents benign overlaps, such as a university and a commercial provider both maintaining maps of the same campus.

The key difference between our approach and traditional hierarchies is the explicit allowance of overlaps. Multiple zones may cover the same physical region, and each can delegate to its own set of servers. This flexibility enables incremental deployment, where new maps can be attached under existing zones without requiring reorganization. Figure 3 shows an example map zone hierarchy with map servers within zones.

While overlaps enable easier integration of new maps, they also introduce challenges for map server discovery. A discovery query must now search across multiple branches of the hierarchy, increasing the complexity of discovery. We elaborate on this and discuss a feasible implementation in § 4.

Why hierarchy?

Instead of organizing as a hierarchy, every map server could register its coverage with a centralized service that allows overlaps. This service could leverage existing spatial databases [36, 43] to store the coverage of each map server simplifying the discovery process. However, this undermines the goals of federation, since new map registrations and updates to coverage would be controlled by a single entity. A peer-to-peer design would avoid centralization but struggle to scale for complex discovery queries. A hierarchical system provides structure, supports federation through delegation, and scales more naturally with growth.

3.1.3 Expressing Coverage

Polygons are the most intuitive representation to express the coverage of map servers and zones, but operations such as intersection checks require complex spatial indexes that are costly to build and query in distributed settings. Furthermore, polygon-based queries are not amenable to distributed caching as they rely on exact spatial boundaries. Small differences in query boundaries can cause cache misses, reducing cache reuse and increasing recomputation.

We can instead express coverage as a collection of primitive shapes. Existing spatial indexing systems like H3 [48] and S2 [24] represent regions using primitive shapes. H3 and S2 decompose the world into hierarchically organized hexagons and squares respectively. The advantage of using spatial indexing systems is that discovery operations can be performed directly on the indices of primitive shapes. Moreover, queries for the same region can be cached using these indices as keys. Precisely expressing an arbitrary region might require a large number of primitive shapes. However, in practice, map coverage boundaries are rarely exact. While it is practical to guarantee that the coverage lies within a boundary, it is often difficult to assert that coverage of a zone or server extends up to every point of that boundary. We can exploit fuzzy boundaries in the real-world to represent regions approximately and bound the number of primitive shapes required to cover them.

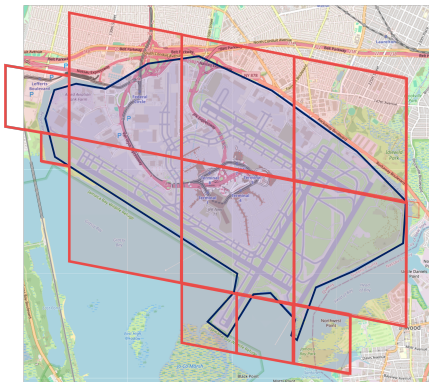


Figure 4 S2 cell covering for a given map boundary.

Unlike squares, hexagons do not exactly subdivide into child hexagons. As a result, the inclusion of children in parents is only approximate in H3, making S2 more suitable for representing an inclusion hierarchy of map zones. Therefore, we use the S2 spatial indexing system to express coverage as a collection of S2 cells, together with an optional altitude above sea-level. Figure 4 shows an example of S2 cells representing a map zone.

3.2 Discovery Query Model

A map server discovery query takes as input a list of S2 cells and optionally, altitude and a list of accepted map zones. It returns the set of map server addresses whose coverage intersects with the input S2 cells. Discovery is limited to a specific altitude, if provided. It is also restricted to map servers registered within the accepted map zones list, if provided.

$$\begin{aligned}&\text{discover(S2 Cells, [altitude], [accepted map zones])} \rightarrow \\ &\qquad [\text{map server}_1, \text{map server}_2, \cdots]\end{aligned} \tag{1}$$

For convenience, an application can represent the search region as a 3D bounding volume, i.e., an arbitrary 2D geographic shape (e.g., a polygon or circle) together with an altitude range. The *OpenFLAME* client library invoked by the application converts the bounding volume to a collection of S2 cells and altitude.

An application that already knows map zones relevant to its context can restrict the discovery process to those zones. For example, a university navigation application can restrict discovery to the university zone. If no list is provided, the discovery query is answered by first recognizing the set of map zones that cover a region followed by identifying all the map servers within these zones that cover the queried region. A consequence of allowing overlaps of map zones is that discovery queries now have to explore all branches of the map zone hierarchy that have coverage over the queried bounding region. Figure 3 shows an example where a query within a university returns maps across zones such as the city waterworks zone, university's own zone, and Google Maps. An advantage of using S2 cells is that the query results can be cached across multiple layers such as client devices, Content Distribution Networks (CDNs), and Internet Service Providers (ISPs). We show in our evaluation (§ 7) that, despite its complexity, the discovery query is manageable due to caching.

Limitations.

The discovery query model we adopt is inherently limited. It does not accept custom search terms and therefore cannot support nuanced queries such as "find only maps of shopping complexes in a city". Importantly, it does not support a ranking criteria to order the discovery results. As a result, querying over a region with many overlapping areas may yield large, unordered result sets. We envision that, once deployed, discovery of maps will evolve much like the Web. While the maintenance and updating of maps must be federated, much like websites on the Web are maintained independently, the discovery of maps can be centralized, analogous to a Web search engine. Centralized search engines would act as curators that crawl regions for available maps, index metadata and the services provided by map servers, and support richer search capabilities. Additionally, mechanisms such as external whitelists, blacklists, and ranking systems (e.g., crowd-sourced voting, auctions) can be employed to further tune discovery results. The discovery mechanism implementation described later in this paper (§ 4) would still be essential to allow such out-of-band mechanisms to discover maps in the first place. We leave the study of nuanced map searches, filtering, and ranking mechanisms to future work.

3.3 Security model

3.3.1 Threat model

Once federated mapping is deployed at scale, it will likely reveal a wide and diverse attack surface. Map servers, for example, are vulnerable to attacks such as denial of service (DoS), reflection, and amplification. We believe that existing methods on DoS protection or anomaly detection could mitigate such attacks. In this work, we choose to focus on a subset of threats unique to *OpenFLAME* that stem from identity spoofing. Identity spoofing in this context refers to an adversary masquerading as a legitimate map server or zone, tricking applications into trusting falsified associations between regions and services. This would lead to threats such as cache poisoning and man-in-the-middle attacks. Identity spoofing may also result in denial of service for spatial applications as they could be bombarded with fake discovery results. As our model relies heavily on caching, and cached records may not always originate from the authoritative source, ensuring the authenticity of the discovery results is crucial.

3.3.2 Chain-of-trust model for spaces

Several existing models provide inspiration for how such identity authentication can be accomplished. For example, the Web relies on Public Key Infrastructure (PKI) [12], where trust anchors are global Certificate Authorities (CAs) that issue digital certificates binding identities to public keys. In contrast, DNSSEC [2] and BGPSEC [32] use hierarchical chain-of-trust systems, where authority is delegated step by step (e.g., root to top-level domain to child domain), and each link in the chain validates the next.

We adopt a hierarchical chain-of-trust model for *OpenFLAME* in which each zone signs all map servers and child zones registered with it. Clients performing discovery have configured trust anchors and validate results by ensuring that the signature chain extends to those anchors. We argue that this model is suitable for spaces as it is easier for new participants to prove their authenticity to a parent zone than to undergo validation from an external CA. For example, it is easier for a student club wanting to host a map of the university to request for validation from the university admins than from a centralized CA. Our data model is

inherently hierarchical, making it natural for a parent zone that references a child to also provide a signed attestation of that child's validity.

Limitations.

Hierarchical chain-of-trust is inherently easier to compromise than a tightly controlled centralized PKI, as careless or malicious child zones may issue faulty signatures compromising the rest of the chain below them. As described in § 3.2, we expect future spatial applications to rely on external curators, filter lists, and ranking mechanisms to select maps in a region. The security mechanism here serves to establish a baseline for identity.

3.3.3 Communication with Map Servers

Once map servers are discovered, clients can authenticate with them using standard mechanisms like password-based login or OAuth [27] with OpenID [45]. As map servers are discovered dynamically, it becomes increasingly important for the client to authenticate the server's identity (e.g., through X.509 certificates [12] or DNSSEC [2]). Subsequent communication between clients and servers to obtain map services can leverage whatever security protocols are most appropriate for the application context. For instance, an image-based localization service might incorporate privacy-preserving features to preserve confidentiality of both the server and the client device [47, 37, 46].

4 Implementation on DNS

4.1 Why DNS?

Existing spatial databases, such as PostGIS [43] and MongoDB [36], can support *OpenFLAME* discovery query (i.e., Query 1) off-the-shelf. They provide datatypes to represent polygons, can index geospatial coordinates, and have fast intersection operators to answer discovery queries. Hierarchy and delegation of zones can be implemented as a collection of spatial databases. However, using such databases would require developing mechanisms for federated and distributed operations, including query routing, response merging, and response caching. We believe that it is better to begin with a design that lends itself to distributed operation from the start.

The data model of *OpenFLAME* map servers and zones described in § 3 is analogous to the DNS model in several ways. First, the map zone is akin to a DNS zone representing an autonomous zone of administration. A link to a map server from a map zone is akin to a DNS record within a DNS zone. Second, DNS has a similar model of hierarchy and delegation for its zones as is described for map zones. Third, the DNSSEC model follows the chain-of-trust mechanism with pre-configured trust anchors similar to the security model described in § 3.3.

Beyond the data model, DNS infrastructure also aligns well with the needs of *OpenFLAME* discovery. First, DNS has pervasive distributed caching across multiple layers such as local caches, ISP resolvers and CDNs which could benefit *OpenFLAME* discovery queries. Second, discovery queries are exclusively read queries and do not need a full-fledged database with transaction processing. Third, map servers and zones are not expected to change very frequently so we do not need low update latencies making DNS suitable.

While the models of *OpenFLAME* and DNS share many similarities, key differences make it non-trivial to adopt DNS directly. First, DNS does not natively support location-based queries or registrations. In § 4.2, we show how regions can be encoded as collections of

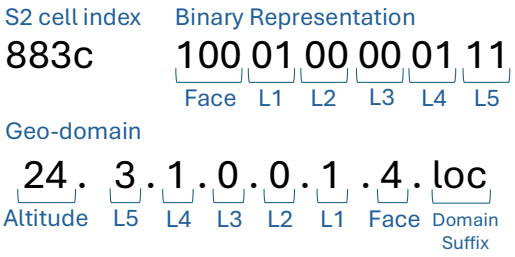


Figure 5 Converting S2 cell index to geo-domain.

domain names compatible with DNS. Second, unlike DNS where each zone is delegated to a single owner, *OpenFLAME* permits overlaps between the coverages of map zones. As a result, discovery queries may need to traverse multiple delegation chains rather than a single path. We introduce additional records (§ 4.3) and query mechanism (§ 4.5) to support such overlaps.

4.2 Geo-domains

Geo-domains are the DNS-compatible addresses representing a physical region. We use the spherical geometry library S2 [24] to generate geo-domains from a bounding region.

S2 projects the Earth's surface onto a perfect mathematical sphere [26]. The sphere is then decomposed into a hierarchy of cells called S2 Cells [25]. Each S2 Cell is a quadrilateral bounded by four spherical geodesic lines—lines along the shortest path on a sphere. The highest level in the S2 hierarchy at level 0 is called a *Face*. Faces divide the Earth into six large quadrilaterals. The top-level faces are recursively subdivided at each level into 4 smaller children. Cell levels range from 0 to 30, and the smallest cells at level 30 have an area of about $1cm^2$. The six top-level cells are numbered from 0 to 5. At each level, the four child cells are numbered from 0 to 3. Each S2 cell has a 64-bit index that is essentially a concatenation of the hierarchical cell numbers in binary form with padding to extend the length to 64 bits. The S2 library also provides the region coverer algorithm—an API that returns the set of S2 cells that approximately cover a given bounding region.

The process of converting an S2 cell index to a geo-domain is shown in Figure 5. The S2 cell index encodes the full hierarchy of the cell up to the top-level face which we convert to a DNS-compatible format. The geo-domain is organized such that the top-level face is at the right end and the smallest cell is at the left end of the domain name. We need a *domain suffix* to represent the root domain of the geo-domain. In our examples, we use `.loc`. The altitude of the S2 cell, rounded up to the nearest integer in feet, is prefixed to the geo-domain. If the altitude is unknown, the letter '*U*' is prefixed instead.

4.3 DNS records

We introduce three new DNS record types to represent map servers, zones, and delegation signers. Although the structures of data in these records are same as the existing DNS records (`NS`, `CNAME`, and `DS` respectively), we introduce new records to support overlaps between zones. The owner names of all of these records are geo-domains.

- `MAPSERVER` record holds the address of a map server. Its structure is the same as a `CNAME` record on traditional DNS that holds the canonical name (or alias) of a domain name.

nameserver: `cmu-nameserver.edu.`

```
$origin 1.0.4.6.4.loc

3.3     IN  TXT { type: MAPZONE, data: cs-ns.univ.edu }
1.1.3   IN  TXT { type: MAPZONE, data: bio-ns.univ.edu }
2.2.4   IN  TXT { type: MAPSERVER, data: admin-maps.univ.edu }
2.2.4   IN  TXT { type: MAPSERVER, data: student-maps.net }
```

Figure 6 Geo-domain DNS records.

Discovery queries request `MAPSERVER` records associated with one or more geo-domains. A geo-domain can have multiple `MAPSERVER` records associated with it to allow overlaps, unlike `CNAME` records [34]. A request for a `MAPSERVER` record informs the resolution system (e.g., DNS recursive resolver) that multiple delegation chains need to be followed to answer this request.

- `MAPZONE` acts like an `NS` record in DNS, i.e., it delegates authority for a sub-zone to the appropriate name servers. Unlike `NS`, however, the same geo-domain may have multiple `MAPZONE` records, each leading to a different delegation chain. While DNS allows multiple `NS` records for the same domain, the underlying assumption is that they all lead to the name servers maintained by the same administrative entity with the same data [34, 35]. However, for *OpenFLAME* discovery queries, resolvers have to follow the delegation chains of all `MAPZONE` records encountered.
- `MAPDS` functions like a `DS` record in DNS, which DNSSEC uses to authenticate the delegation from a parent zone to a child zone [3]. The key difference is that, unlike `DS`, a single geo-domain can hold multiple `MAPDS` records, each corresponding to a different delegation chain.

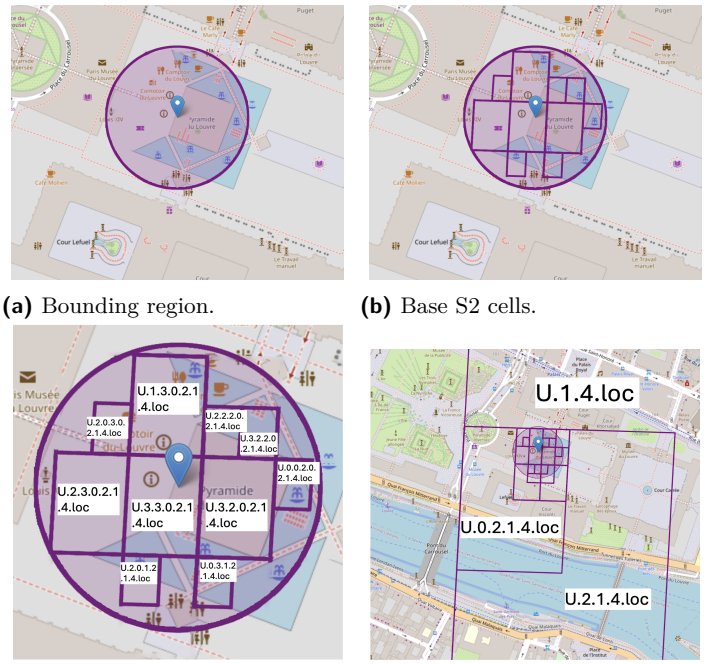
To ensure backwards compatibility with existing DNS servers, we wrap the above records in `TXT` records which can hold arbitrary data and is universally supported by all DNS implementations. As a result, map zones can be hosted on standard DNS nameservers without any changes. Figure 6 shows an example zone file in *OpenFLAME*. It has `MAPZONE` records pointing to name servers maintained by separate departments. It also has `MAPSERVER` records pointing to map servers hosting the map of the university admin building. We implement map servers as web servers that implement map services such as localization, routing, and geocoding (§ 5).

4.4 Map registration workflow

To understand the map registration workflow, let us consider an example of hosting an airport's map on *OpenFLAME*; for clarity, we omit signature and authenticated delegation records here and defer their discussion to § 4.6 on security.

First, a polygon is roughly drawn around the map region. S2's region-coverer algorithm is used to generate S2 cells covering the polygon. In Figure 4, the blue polygon represents the map's boundary and each red square represents an S2 cell. The covering does not perfectly align with the blue polygon marking the map boundary. This is acceptable since maps on *OpenFLAME* have fuzzy boundaries. The altitude of the map is mentioned or it is marked as unknown.

For each S2 cell, a geo-domain is generated as shown in Figure 5. A `MAPZONE` record for each geo-domain is registered at the parent zone's DNS nameserver (e.g., the nameserver



(a) Bounding region. **(b)** Base S2 cells.

(c) Base geo-domains. **(d)** Parent geo-domains.

Figure 7 Generating geo-domains for discovery.

hosting the city map zone), enabling queries within the airport to be delegated to the airport's map zone. These `MAPZONE` records point to the airport's DNS nameserver. If altitude information is available, an additional set of geo-domains is registered with the altitude marked as unknown (using the prefix 'U'). This allows the zone to remain discoverable even for clients that lack altitude data.

Note that the airport is not required to register under a single, globally sanctioned parent zone. Instead, it may choose any parent zone whose spatial coverage includes the airport region. For example, if a city-level map zone declines to allow registration of the airport's map zone for any reason, the airport can register under an alternative parent zone operated by another organization. As long as the parent zone advertises coverage over the relevant region, the airport's map remains discoverable through *OpenFLAME*'s discovery mechanism.

Once the airport zone is established, individual map servers can register `MAPSERVER` records in the airport's DNS server pointing to the corresponding servers providing map services. For instance, each terminal may operate its own map server. The airport zone may also delegate portions of its coverage to sub-zones by publishing `MAPZONE` records in its nameserver.

4.5 Discovery query workflow

As described in § 3.2, an application expresses search region as a bounding volume (i.e., a 2D geographic region with altitude). The *OpenFLAME* client first converts the request to a discovery query of the form (1) and then represents it as geo-domains. Figure 7 shows our technique for converting a bounding region to geo-domains.

1. We use S2's interior region coverer algorithm to get a set of S2 cells that cover the 2D geographic region. We call these the *base S2 cells* (Figure 7b).

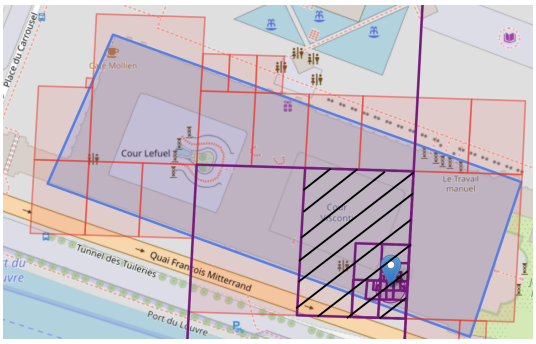


Figure 8 Intersection of **queried** and **registered** geo-domains.

2. For each base S2 cell, its corresponding geo-domain is generated (Figure 7c). We call these geo-domains the *base geo-domains*.
3. For each base geo-domain, we generate all the parent domains by removing sub-domains sequentially from the left. We retain the altitude. For example, the parents of the domain `U.1.3.5.loc` are `U.3.5.loc`, and `U.5.loc` (Figure 7d). Querying parents is essential to discover maps whose coverages are larger than the queried region. Optionally, we also add the children of base geo-domains upto a predefined level.
4. The set of geo-domains to query includes all the base geo-domains, and their parents with duplicates removed (and optionally, their children). Optionally, a copy of the same geo-domains but with altitude marked as unknown is also queried to discover map servers not registered with a known altitude. Queries for `MAPSERVER` records of all these geo-domains are made in parallel to get a list of map servers for the queried region.

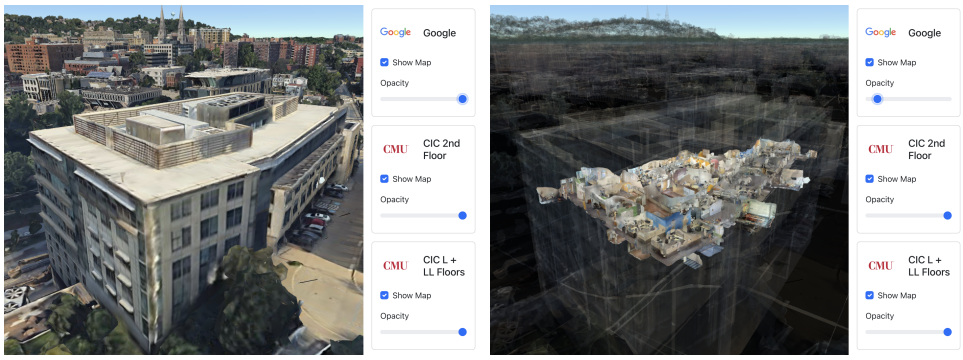
Figure 8 illustrates a discovery query. Red S2 cells show the geo-domains registered for a map server serving a museum, while purple S2 cells show the geo-domains generated and queried for the input region (the small purple circle). The intersection of the registered and queried geo-domains, shown with black stripes, ensures that the address of the map server is returned by the DNS when the discovery query is made.

Caching.

In every discovery cycle, the *OpenFLAME* client typically queries about 40 geo-domains (about 10 base geo-domains and 30 parent geo-domains. See § 7 for details). The ubiquitous caching mechanisms in DNS make such a large number of queries feasible. Most of the geo-domain queries are answered by the local DNS cache and never leave the device. This is because some of the top-level parent geo-domains (Figure 7d) rarely change as the queries from a client are usually limited to a small region in the real world.

Fan-out queries.

In traditional DNS resolution, a query, such as one for a `CNAME` record, proceeds by following a delegation chain through `NS` records. If multiple `NS` records are present, the resolver arbitrarily selects one and follows only that chain, since all entries are assumed to lead to equivalent answers. In *OpenFLAME*, this assumption does not hold because overlapping zones may create multiple valid delegation chains for the same geo-domain. To correctly resolve a `MAPSERVER` record, the resolver must therefore traverse all `MAPZONE` records in parallel, exploring every chain of delegation rather than selecting one. In our implementation,



(a) Outdoor view of a building from Google Maps.

(b) An indoor map of offices hosted on private servers.

Figure 9 Map visualization application built on *OpenFLAME*.

to maintain backwards compatibility, we leave standard DNS resolvers unmodified. Instead, the client issues `TXT` queries for the requested geo-domains. It then queries name servers in each `MAPZONE` record (wrapped in a `TXT` record) to get `MAPSERVER` records.

4.6 Security/Validation

The security model for spaces described in § 3.3 follows a chain-of-trust approach similar to DNSSEC. In DNSSEC, each record is signed using an `RRSIG` record, and these signatures are validated against public keys stored in `DNSKEY` records. Delegation between zones is authenticated using `DS` records, which establish trust from a parent to a child zone.

Our implementation reuses DNSSEC's `DNSKEY` and `RRSIG` records as well as its standard zone-signing protocols. The key difference lies in how delegation is handled. To support multiple delegation chains for the same geo-domain, we introduce a new record type, `MAPDS`. A geo-domain may have several `MAPDS` records, each corresponding to a different delegation chain. As a result, authenticating a `MAPSERVER` or `MAPZONE` record may require traversing all relevant delegation chains.

5 Building Map Services

In this section, we discuss examples of implementing map services on the *OpenFLAME* infrastructure. We consider five key services: tile server (i.e., map visualization, § 5.1), forward and reverse geocode(§ 5.2), routing (§ 5.3), and localization (§ 5.4). While *OpenFLAME* can support arbitrary services, we highlight these five because they span the spectrum of inter-map interactions: tile server and reverse geocoding (location-to-address mapping) do not require interaction with other maps, forward geocoding (address-to-location search) relies on a root map to bootstrap the search, and routing and localization require combining results across multiple maps. See Appendix C for a discussion on combining these services across maps that use different coordinate systems.

5.1 Tile Server

The tile service serves the application with a visual representation of its map data. The tile service on our map servers serves 3D scans of indoors as meshes, along with a graph representing map data. We build an interactive map application that uses *OpenFLAME*

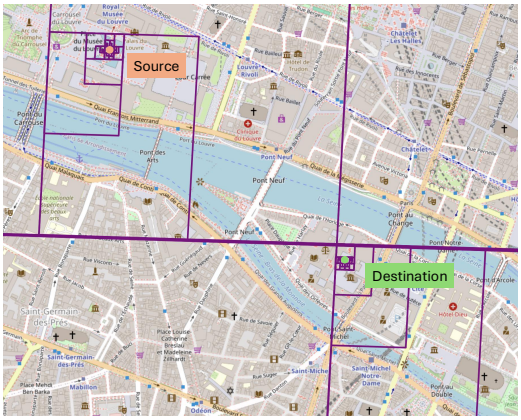


Figure 10 Geo-domains queried for routing

to discover map servers hosted in the user's view region, downloads tiles from them, and renders these 3D tiles alongside others for the same region. Figure 9 shows two screenshots from our application. Google Maps is registered as a server with global coverage and its 3D map tiles service [23] provides outdoor 3D scans of buildings (Figure 9a). Figure 9b shows indoor scans of offices within the building served by private university map servers that are exposed by reducing Google tiles opacity. These tiles are only visible to users with university credentials.

5.2 Forward and Reverse Geocode

The process of converting a text-based address to a location on the map is called forward geocoding [22]. In a centralized map provider, the text-based addresses of map nodes are indexed against their geolocations. So geocode involves querying this indexed database. In *OpenFLAME*, we need a *root map* with global coverage to bootstrap the geocoding process. In our case, large world-map providers such as OpenStreetMap, Google or Apple maps serve as root maps. Given a text string of a hierarchical address, the client first uses the geocode service of a root map to get the coarse location of a part of the address. The client then discovers map servers for this coarse location. It requests geocoding service from each of these map servers which search in their maps for the exact address. For example, let us consider the address "Mona Lisa Painting, Louvre Museum, Paris, France". While OpenStreetMap's data might not contain the location of the painting in the museum, it will give the location of the Louvre museum. *OpenFLAME* client can then discover the Louvre map server and search for the location of the painting within that map server.

The service that coverts a geolocation to a map node is called reverse geocode. Given a geographic location, the *OpenFLAME* client uses the discovery system to find all the map providers in that location. Then it requests the reverse geocode service from each discovered map server and returns to the client the node that is closest to the requested geolocation.

5.3 Routing

Routing refers to calculating a path from source to destination with optimization objectives such as minimizing travel duration, distance, or toll price. Routing in a centralized map is performed by running shortest path algorithms on the graph representation of the map [20, 4]. We present a simple implementation for routing on federated maps that we find is sufficient

for most common cases. Given the source and destination addresses, we first use the geocode service to find their approximate geolocations. We then use the discovery system to find map providers near the source and destination. Figure 10 shows the geo-domains queried. Querying smaller geo-domains near the source and destination locations discovers smaller and possibly more detailed maps. The client also discovers large maps that cover the region between the source and destination. The client uses a large map that spans both the source and destination to get an approximate route from the source to the destination. It then contacts smaller maps near the source and destination to refine these routes.

Let us reconsider the campus navigation application from § 2. The *OpenFLAME* client would discover that Google Maps spans both the source—the street that the user is currently in, and the destination—the university premises (even though Google Maps only has coarse information at the destination). The client then requests an approximate route from Google Maps that would lead the user to the university campus entrance. It then requests the university's map server for a route from the university entrance to the professor's office. The client then stitches these two routes together to obtain a complete route from the street to the office.

This simple algorithm works for the common case where world-scale maps such Google Maps is sufficient to navigate the user outdoors, switching to other smaller maps only when indoors near the source or destination. However, this does not work for situations where the optimal path passes through smaller maps. Small detailed maps along the path are never discovered. There is a trade-off between the number of map servers contacted and the optimality of the final route. In § 7.3, we evaluate the optimality of our simple routing algorithm compared to a centralized algorithm. We leave the exploration of better routing algorithms to future work.

5.4 Localization

Localization is the service that informs the application the position and orientation or the pose of a device with respect to a map. Centralized maps are set against the system of latitudes and longitudes so spatial applications rely on geo-positioning systems such as GPS, WiFi, and cell towers to position the device with respect to the map. Maps in *OpenFLAME* can be indoors (where GPS does not work reliably) and can be laid out in separate 2D/3D coordinate system of their own. To ensure devices can localize themselves within such maps, the map servers will have to provide their own localization service.

Localization can be performed using various technologies including Ultra-Wideband (UWB) beacon, Bluetooth, Wi-Fi, and image-based methods. Image-based localization is a well-studied problem [31, 40, 30, 39]. Our prior work [5] demonstrates image-based 6DoF localization across federated 3D maps. It uses a combination of local trajectories from device VIO sensors and remote trajectories from VPS (Visual Positioning System) servers to select the best map. It also shows how to stitch localization results across maps without exposing internal map details. Here, we focus on how localization integrates with the discovery layer of *OpenFLAME*. Figure 11 illustrates the workflow. Generating geo-domains, making DNS queries, and sending images to all servers is costly in both computation and network overhead. We avoid repeating this process by exploiting temporal locality. That is, a device typically remains within the same map for a contiguous period of time. Thus, we can repeatedly send cues to the same server (i.e., `activeServer`) until localization quality degrades, at which point rediscovery is triggered. In § 7.2, we evaluate the client's ability to switch between map servers.

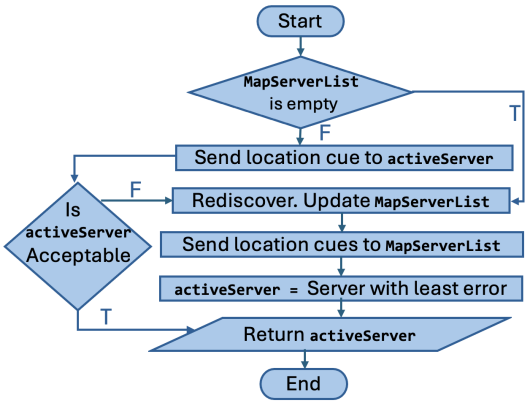


Figure 11 Localization and discovery flowchart.

6 Implementation

Client library: We implement *OpenFLAME* as an `npm`[42] library, bundling it for browsers to support platform-agnostic web applications. Since browsers lack direct system calls for DNS queries, we use DNS over HTTPS (DoH)[28], supported in the latest BIND9 [6] DNS server implementation.

DNS: Our DNS server uses the BIND9 implementation. For evaluation, we run our DNS server on a machine with Intel Core i9-9820X CPU. We also provide a web-based tool to automate the process of generating geo-domains.[3]

Localization: We implement the federated image-based localization service as described in prior work [5]. We run localization on a machine with Intel Core i9-13900K CPU and NVIDIA GeForce RTX 4090 GPU.

Routing and geocoding: Within map servers, we use Nominatim [41] for geocoding and Open Source Routing Machine (OSRM) [33] for routing.

Interactive map: The web application is implemented using CesiumJS [9], a JavaScript library for creating 3D maps. Outdoor map tiles are streamed using the Google Photorealistic 3D Tiles API [23]. We create indoor scans using Polycam [44] and convert them to map tiles using Cesium Ion [8]. These indoor map tiles are served from a Django [19] web server access-controlled using oAuth [27] and OIDC [45] to restrict tile service to specific logins.

7 Evaluation

We evaluate the DNS-based discovery system in § 7.1. We show that most of the DNS queries in *OpenFLAME* are cache hits and that a standard bind9 DNS deployment can support a large number of users despite discovery involving a large number of DNS queries. In § 7.2, § 7.3 and § 7.4 we discuss the performance of localization, routing, and reverse geocode on federated maps and compare them with their centralized equivalents.

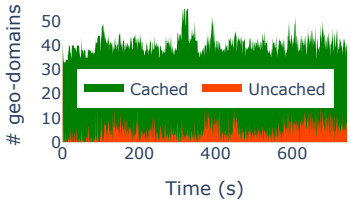
7.1 Discovery

To evaluate the discovery phase, location traces with latitude, longitude, and 95% confidence radius were collected by the authors. These traces are representative of the movement of a

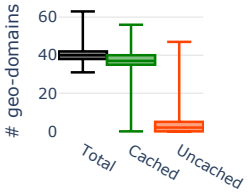
[3] `https://www.open-flame.com/`

Figure 12 Examples of location traces collected.

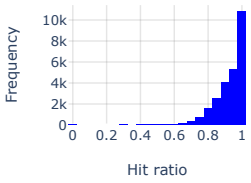
typical device that may run a location-based application on *OpenFLAME*. Figure 12 shows a few examples of the traces collected. The blue lines show the latitudes and longitudes and the red circles show the 95% confidence radius. Note that the error is higher indoors than outdoors, as expected. We then ran the discovery phase on each of these traces.



(a) Number of geo-domains per discovery query over one trace.



(b) Geodomains queried over all traces.



(c) Distribution of cache DNS hit ratios.

Figure 13 Geodomain query statistics in *OpenFLAME*.

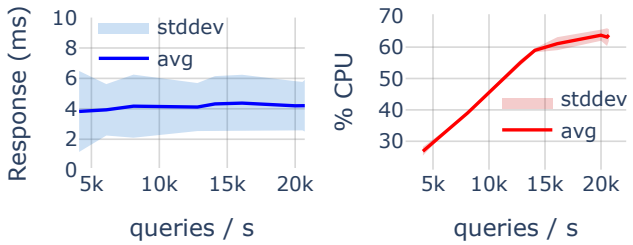
Figure 13a shows the number of geo-domains queried for one of the traces. The number of geo-domains queried is consistently high throughout the trace with a median of 36. However, the number of uncached geo-domains that have to be queried from the DNS server is small with a median of 4. Most of the geo-domain queries are answered by the local cache as the device does not arbitrarily jump to radically different locations and query different geo-domains.

Figure 13b shows the box plot of the total number of geo-domains queried across all records in all traces. Notice that most of the DNS requests are resolved from the cache. The long whiskers for both the cached and uncached box plots are because of the first *OpenFLAME* query in the session. In the first query, all geo-domains need to be queried as the cache is empty. Figure 13c shows that the DNS cache hit ratio is close to 1 for a large number of *OpenFLAME* queries.

We used `dnsperf` [14] to benchmark our DNS server. The DNS requests were generated on a machine in the same local network as the DNS server. Figure 14a shows that even at 20,000 DNS queries per second, the server response latency stays at 4 ms. This means that a single DNS server, without complex infrastructure set up, can support thousands of *OpenFLAME* clients resolve geo-domain queries.

7.2 Localization

The localization service needs to work in tandem with the discovery phase as the user moves through different maps. Figure 15 shows the network activity on an *OpenFLAME* client



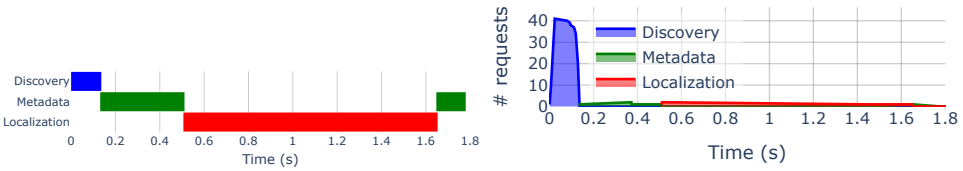
(a) Response latency. **(b)** CPU Usage.

Figure 14 DNS server capacity.

running the localization service. Figure 15a shows the time that the client spends in different phases in one cycle of queries including discovery and localization. Each horizontal bar represents the time from when the request was sent to when the response was received, including the time the client spent waiting for the response. Metadata refers to the negotiation of localization technologies. The client waits over 1 second for the localization result from the map servers. Note that other localization technologies could require far less time for the localization step of the process. In our AR application, while it waits for localization results, we continue to render AR content using the local tracking implemented by WebXR. We used the localization result to get an initial pose estimate and to correct drifts in local tracking. Therefore, the waiting time does not significantly affect the user experience.

Figure 15b show the number of requests sent in each phase. About 40 geo-domains are queried in the discovery phase as evidenced by the large number of network requests sent in the discovery phase. Despite a large number of requests, the time spent in the discovery phase is much shorter than in the localization phase (Figure 15a). This is because *OpenFLAME* makes all DNS requests in parallel and the DNS server responds in a few tens of milliseconds. To capture the worst case for the number of requests sent in the discovery phase, we configure our DNS servers to prevent caching of NXDOMAIN results (i.e., negative results). However, as we show in § 7.1, only the first request involves a large number of DNS queries, after which most geo-domains are cached.

As described in § 5.4, *OpenFLAME* does not run the discovery phase repeatedly but rather maintains an `activeServer` (Figure 11). To verify that *OpenFLAME* returns the correct map server without having to run rediscovery every time, we set up 4 map servers, each covering a different indoor region in a university building. The regions were close enough that GPS location could not reliably distinguish between the 4 regions indoors. As a result, every time rediscovery is triggered, the addresses of all 4 map servers are returned and images are sent to all of them. The dotted lines in Figure 17 show the confidence of the map servers normalized between 0 and 1 as a reference. *OpenFLAME* does not contact all



(a) Duration of network requests. **(b)** Number of network requests.

Figure 15 Network activity of localization service.

servers every time, and therefore does not have access to this data but rather uses local VIO poses to estimate 'client error'. The colored dots show the selected `activeServer`. We see that in most iterations, the `activeServer` is the server with the highest server confidence. The stars in the figure show that rediscovery roughly corresponds to the times when the confidence scores for the `activeServer` dips. Two consecutive rediscoveries are triggered when the confidence scores are low for all servers in a region where no map server has good coverage.

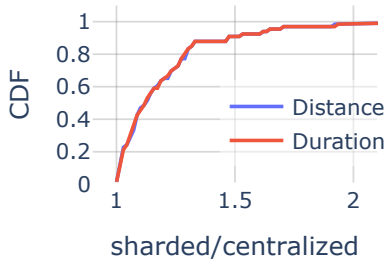
7.3 Routing

We evaluate our implementation of the routing service (§ 5.3) on an OpenStreetMap of a city. The routing service is run on two versions of the map—a centralized version which retains the original form of the city map, and a sharded version where we split the map into a *root city map* and smaller sub-maps. The smaller sub-maps consist of universities and some neighborhoods cut out of the city. We generated geo-domains for these sharded regions and registered them on our DNS server. These DNS records point to map servers providing routing service for each shard. We use the Open Source Routing Machine (OSRM) [33] on the map servers to calculate optimal routes within maps. A routing query in the centralized version is made to the map server running OSRM on the whole city's map. In the sharded version, routing involves discovering relevant map servers using DNS queries and then requesting the map servers to run OSRM on their portion of the map.

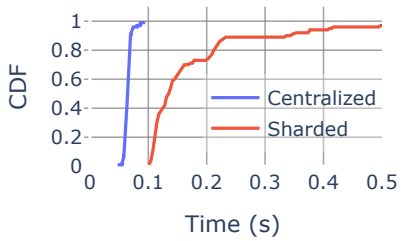
Figure 16a shows the ratio of the travel distance along the path returned by sharded and centralized routing. A ratio of 1 indicates that the distances are the same, while higher ratios show the extent of sub-optimality of sharded routing. The median ratio is 1.12 and the 90$^{\text{th}}$ percentile is 1.47. Figure 16b shows the CDF of the times taken for the completion of routing queries on the sharded and centralized models. These times include both the discovery and route calculation times. At the median, a sharded routing query takes about twice as long as a centralized query.

7.4 Reverse Geocode

We evaluate our implementation of the reverse geocode service using the same setup of centralized and sharded maps as described in § 7.3. Figure 18 shows the CDFs of the time taken for reverse geocode queries on sharded and centralized maps. It includes both discovery and map data query time. At median, the ratio of sharded to centralized time is 1.67.



(a) Routing optimality ratios.



(b) Routing time taken.

Figure 16 Routing performance on Sharded maps.

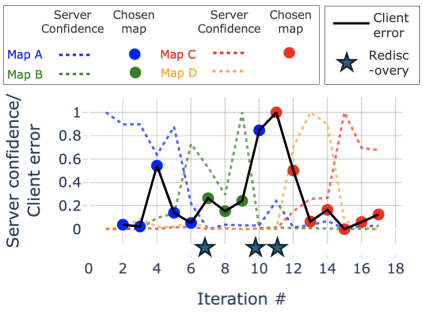


Figure 17 Map Selection.

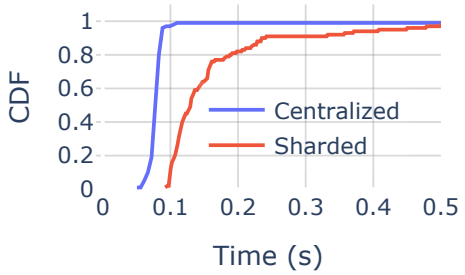


Figure 18 Reverse Geocode performance.

8 Related work

Federated mapping.

MapCruncher [16] recognized the need to allow interoperability of distributed geographic data, focusing on layering interactive map data (called 'mashups') from different sources. MapSynthesizer [15] built an application on top of MapCruncher that could discover and render tiles from distributed sources. They do not go into the details of how map data from different sources can be organized and discovered as their focus is on interactive map tiles. Our approach is to treat maps as an abstraction, and build an infrastructure to organize and discover them so they can expose any generic service (including visualization) to applications.

DNS-based discovery.

There have been some proposals in the past that have used the DNS for discovery of services. DNS-based Service Discovery (DNS-SD) [10] (eg. Apple Bonjour [1]) uses the DNS to discover services of a given type under a given domain. In these systems, discovery is not based on geographic location but within a local network. DNS LOC records [13] store latitude and longitude as part of record data. These records only serve as metadata associated with a domain name and do not assist with location-based discovery. Several proposals have been made to extend the DNS to support geographic location-based queries, specifically for Vehicular Ad-hoc networks (VANETs) [29, 17, 18, 38]. However, these proposals require altering DNS implementation and were not adopted. Recent work by Gibb et al [21] introduces location-aware DNS queries by leveraging the hierarchy in civic addresses of locations. It does not explore how to generate these domains to discover nearby services.

9 Conclusion

In this paper we present the design and implementation of *OpenFLAME*, a federated mapping system for spatial applications. *OpenFLAME* organizes the world into smaller maps hosted on map servers maintained by disparate parties. It can incorporate private indoor maps, is scalable, and has a low barrier to entry. We implement a DNS-based map server discovery system so we can reuse existing DNS infrastructure and caching mechanisms. We also implement location-based services, including map visualization, geocoding, routing, and localization, on federated maps. We believe that a federated mapping system is essential for future spatial applications and hope that our paper acts as an impetus for the research community to start democratizing maps.

References

1. Apple. Apple Bonjour. https://developer.apple.com/bonjour/, 2025. Accessed: 2024-09-01.
2. Roy Arends, Rob Austein, Matt Larson, Dan Massey, and Scott Rose. Rfc 4033: Dns security introduction and requirements. RFC 4033, 2005. URL: https://datatracker.ietf.org/doc/html/rfc4033.
3. Roy Arends, Rob Austein, Matt Larson, Dan Massey, and Scott Rose. Rfc 4034: Resource records for the dns security extensions. RFC 4034, 2005. URL: https://datatracker.ietf.org/doc/html/rfc4034.
4. Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. *Route Planning in Transportation Networks*, pages 19–80. Springer International Publishing, Cham, 2016. doi:10.1007/978-3-319-49487-6_2.
5. Sagar Bharadwaj, Harrison Williams, Luke Wang, Michael Liang, Tao Jin, Srinivasan Seshan, and Anthony Rowe. Openflame: Federated visual positioning system to enable large-scale augmented reality applications. In *2025 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. IEEE, 2025.
6. BIND. BIND. https://www.isc.org/bind/, 2025. Accessed: 2024-09-01.
7. University of Michigan Center for Sustainable Systems. Commercial Buildings Factsheet. https://css.umich.edu/publications/factsheets/built-environment/commercial-buildings-factsheet, 2025. Accessed: 2024-09-01.
8. Cesium. Cesium ion. https://cesium.com/platform/cesium-ion/, Jun 2025.
9. Cesium. Cesiumjs. https://cesium.com/platform/cesiumjs/, Jun 2025.
10. S Cheshire and M Krochmal. RFC6763: DNS-Based Service Discovery. https://www.rfc-editor.org/rfc/rfc6763.html, 2013.
11. David D Clark, John Wroclawski, Karen R Sollins, and Robert Braden. Tussle in cyberspace: defining tomorrow's internet. In *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 347–356, USA, 2002. Association for Computing Machinery.
12. David Cooper, Stefan Santesson, Stephen Farrell, Sharon Boeyen, Russ Housley, and Tim Polk. Rfc 5280: Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile. RFC 5280, 2008. URL: https://datatracker.ietf.org/doc/html/rfc5280.
13. C Davis, P Vixie, T Goodwin, and I Dickinson. RFC6762: A Means for Expressing Location Information in the Domain Name System. https://datatracker.ietf.org/doc/html/rfc1876, 1996.
14. DNS-OARC. dnsperf. https://github.com/DNS-OARC/dnsperf, 2025. Accessed: 2024-09-01.
15. Miguel Elias, Jeremy Elson, Danyel Fisher, and Jon Howell. Do i live in a flood basin? synthesizing ten thousand maps. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 255–264, 2008.
16. Jeremy Elson, Jon Howell, and John R Douceur. Mapcruncher: integrating the world's geographic information. *ACM SIGOPS Operating Systems Review*, 41(2):50–59, 2007.
17. Tiago Fioreze and Geert Heijenk. Extending dns to support geocasting towards vanets: A proposal. In *2010 IEEE Vehicular Networking Conference*, pages 271–277, Jersey City, New Jersey, USA, 2010. IEEE.
18. Tiago Fioreze and Geert Heijenk. Extending the domain name system (dns) to provide geographical addressing towards vehicular ad-hoc networks (vanets). In *2011 IEEE Vehicular Networking Conference (VNC)*, pages 70–77, Amsterdam, Netherlands, 2011. IEEE, IEEE.
19. Django Software Foundation. django. https://www.djangoproject.com/, Jun 2025.
20. Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact routing in large road networks using contraction hierarchies. *Transportation Science*, 46(3):388–404, August 2012. doi:10.1287/trsc.1110.0401.

21 Ryan Gibb, Anil Madhavapeddy, and Jon Crowcroft. Where on earth is the spatial name system? In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*, pages 79–86, Cambridge, Massachusetts, USA, 2023. ACM.

22 Daniel W. Goldberg, John P. Wilson, and Craig A. Knoblock. From text to geographic coordinates: The current state of geocoding. *Urisa Journal*, 19:33, 2007. URL: `https://api.semanticscholar.org/CorpusID:5517082`.

23 Google. Photorealistic 3D Tiles. `https://developers.google.com/maps/documentation/tile/3d-tiles`, 2025. Accessed: 2024-09-01.

24 Google. S2 library. `http://s2geometry.io/`, 2025. Accessed: 2024-09-01.

25 Google. S2 library - Cells. `http://s2geometry.io/devguide/s2cell_hierarchy`, 2025. Accessed: 2024-09-01.

26 Google. S2 library - Overview. `http://s2geometry.io/about/overview`, 2025.

27 Dick Hardt. Rfc 6749: The oauth 2.0 authorization framework. RFC 6749, 2012. URL: `https://datatracker.ietf.org/doc/html/rfc6749`.

28 P Hoffman and P McManus. RFC8484: DNS queries over HTTPS (DoH). `https://datatracker.ietf.org/doc/html/rfc8484`, 2018.

29 Tomasz Imieliński and Julio C Navas. Gps-based geographic addressing, routing, and resource discovery. *Communications of the ACM*, 42(4):86–92, 1999.

30 Alex Kendall, Matthew Grimes, and Roberto Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *Proceedings of the IEEE international conference on computer vision*, pages 2938–2946, Boston, Massachusetts, USA, 2015. IEEE.

31 Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In *2007 6th IEEE and ACM international symposium on mixed and augmented reality*, pages 225–234, Nara-Ken New Public Hall, Nara, Japan, 2007. IEEE, IEEE.

32 Matt Lepinski and Kotikalapudi Sriram. Rfc 8205: Bgpsec protocol specification. RFC 8205, 2017. URL: `https://datatracker.ietf.org/doc/html/rfc8205`.

33 Dennis Luxen and Christian Vetter. Real-time routing with openstreetmap data. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '11, pages 513–516, New York, NY, USA, 2011. ACM. URL: `http://doi.acm.org/10.1145/2093973.2094062`, `doi:10.1145/2093973.2094062`.

34 Paul V Mockapetris. RFC1034: Domain names-concepts and facilities, 1987.

35 Paul V Mockapetris. RFC1035: Domain Names - Implementation and Specification, 1987.

36 MongoDB. MongoDB Geospatial Queries. `https://www.mongodb.com/docs/manual/geospatial-queries/`, 2024.

37 Heejoon Moon, Chunghwan Lee, and Je Hyeong Hong. Efficient privacy-preserving visual localization using 3d ray clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9773–9783, June 2024.

38 Daniel Moscoviter, Mozhdeh Gholibeigi, Bernd Meijerink, Ruben Kooijman, Paul Krijger, and Geert Heijenk. Improving spatial indexing and searching for location-based dns queries. In *Wired/Wireless Internet Communications: 14th IFIP WG 6.2 International Conference, WWIC 2016, Thessaloniki, Greece, May 25-27, 2016, Proceedings 14*, pages 187–198, Thessaloniki, Greece, 2016. Springer, Springer.

39 Anastasios I Mourikis and Stergios I Roumeliotis. A multi-state constraint kalman filter for vision-aided inertial navigation. In *Proceedings 2007 IEEE international conference on robotics and automation*, pages 3565–3572. IEEE, 2007.

40 Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015.

41 Nominatim. Nominatim. `https://nominatim.org/`, 2025. Accessed: 2025-01-14.

42 npm. npm. `https://www.npmjs.com/`, 2025.

43 PostGIS PSC & OSGeo. PostGIS. `https://postgis.net/`, 2024. Accessed: 2025-01-14.

44 Polycam. Polycam. `https://poly.cam/`, 2025. Accessed: 2024-09-01.

804 **45** Nat Sakimura, John Bradley, Mike Jones, Breno de Medeiros, and Chuck Mortimore. Openid
805 connect core 1.0, 2014. URL: `https://openid.net/specs/openid-connect-core-1_0.html`.
806 **46** Mikiya Shibuya, Shinya Sumikura, and Ken Sakurada. Privacy preserving visual slam. In
807 *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020,*
808 *Proceedings, Part XXII 16*, pages 102–118. Springer, 2020.
809 **47** Pablo Speciale, Johannes L Schonberger, Sudipta N Sinha, and Marc Pollefeys. Privacy
810 preserving image queries for camera localization. In *Proceedings of the IEEE/CVF International*
811 *Conference on Computer Vision*, pages 1486–1496, 2019.
812 **48** Uber. H3. `https://h3geo.org/`, 2025. Accessed: 2024-09-01.

	Content -centric	Application -centric	Map -centric
Scalability	✗	✓	✓
Incremental deployability	✗	✗	✓
Space ownership Enforcement	Depends on implementation of the discovery system		
Delegability			
Low barrier for map creators	Depends on implementation of location-based services		
Low expectation from clients			

Table 1 Design choices vs. system characteristics.

A Design space exploration

A spatial naming system associates locations with *names*. In the paper we choose *maps* (i.e., labels associated with locations and relationships between labels) as names. However, there are alternate choices for names that can lead to a different system design.

Content-centric design – the naming system associates locations with content hosted at those locations. For example, at a supermarket, the naming system might discover the products sold at the supermarket. We believe that in future spatial applications, content will be dynamically generated at a high rate (as we see on Web applications today). Content-centric design will be bottle-necked at the naming system which is detrimental to spatial applications.

Application-centric design – the naming system associates locations with applications hosted at those locations. For example, at a supermarket, the naming system might discover the product search and navigation application. This is analogous to the Web where names (eg. `google.com`) refer to applications (eg. Web search). The applications maintain all of their content making the naming system independent of the amount of content generated, thereby removing a bottleneck from the system. As the content is spatial, applications need to maintain pointers maps that can provide context about the location of their content with respect to the real world. For example, to navigate users to their products of interest, the application needs a localization service that can determine the position of the user within the store. This inhibits incremental deployability of services. For example, consider the supermarket sharding its localization service to provide it separately for the clothing and electronics section, while decommissioning their store wide service. While this enhances ease of maintenance for the store, it breaks applications that still point to the older store wide service.

Map-centric design – The naming system associates locations with maps and the services provided on these maps. Related maps can be grouped into separate *zones* and maintained autonomously, just as physical regions are maintained by disparate entities. This system is independent of content hosted against maps ensuring scalability. It also affords incremental deployability. Applications author their content with respect to map elements, use standardized interfaces to obtain map services and are agnostic to which machines are serving them. This is the design we choose for *OpenFLAME*. Table 1 shows the matrix of system characteristics against design choices.

Figure 19 AR indoor navigation application built on *OpenFLAME*.

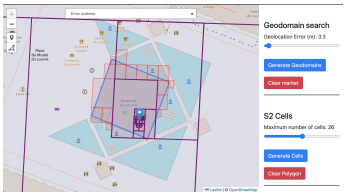


Figure 20 Geo-domain Explorer.

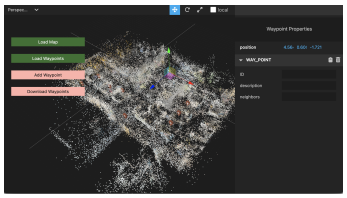


Figure 21 3D map creator.

	Geo-based	Bound	Unbound
Geocode	✓	✓	✓
Reverse-geocode	✓	✗	✗
Routing	✓	✓	✓
Localization	✓	✓	✓
Tile rendering	✓	✓	✓

Table 2 Map services that can be provided on different kinds of maps. ✓ shows the service can be provided on the map and its results can be combined with other maps if needed. ✓ shows the service can be provided on the map but its results cannot be combined with other maps. ✗ means the service cannot be provided on the map.

B Application and Tools

Figure 19 shows a screenshot of the 3D indoor AR navigation application built on *OpenFLAME*. The Geo-domain Explorer tool (Figure 20) automates the process of generating DNS records for registering a map server or zone with the DNS. We have hosted it on `https://www.open-flame.com/`. The Waypoint Tagger tool (Figure 21) helps map creators tag map nodes and ways on their 3D scans. It also exports the map to be used with a map server.

C Map Services on Heterogeneous Maps

Map services need to support a heterogeneous set of maps using different coordinate systems. We call maps laid out in the global geographic system *geo-based* maps. *Local* maps (maps in their own local coordinate system) can be further classified based on how they label areas shared with surrounding maps. If a map uses the same labels for shared areas as the other maps around them, they can be explicitly stitched together, so we call them *bound* maps. Otherwise we call them *unbound*. The discovery system is agnostic to this heterogeneity and discovers all maps and services for a given location.

Table 2 shows the different services that can be supported by each map type. Geo-based maps can support all services. Location-to-address conversion (reverse-geocoding) cannot be supported on local maps as they do not have a notion of latitudes and longitudes. Unbound maps can support routing within the map, but the routes calculated in such maps cannot be combined with paths from other maps as there are no common nodes.