BONVOYAGE

From Bilbao to Oslo, intermodal mobility solutions, interfaces and applications for people and goods, supported by an innovative communication network

Research and Innovation Action GA 635867

Deliverable D3.3:

Travel-centric and participatory sensing services.

Deliverable Type:	R
Deliverable Number:	D3.3
Contractual Date of Delivery to the EU:	31.10.2017
Actual Date of Delivery to the EU:	31.10.2017
Title of Deliverable:	Travel-centric and participatory sensing services.
Work package contributing to the Deliverable:	WP3
Dissemination Level:	PU
Editor:	Andrea Detti [CNIT]
Author(s):	Nicola Blefari [CNIT], Pietro Boccadoro [CNIT], Giulio Rossi [CNIT], Giulio Rossi [CNIT],Riccardo Paolillo [CNIT], Michele Orru [CNIT], Luigi Alfredo Grieco [CNIT], Mauro Losciale [CNIT], Giuseppe Piro [CNIT]
Internal Reviewer(s):	Giuseppe Tropea [CNIT]
Abstract:	This deliverable describes the exploitation of Internames for travel-centric and participaroty sensing services. Exemplary applications are the implementation of ITS discovery services and the dissemination of DATEX II real-time information.
Keyword List:	BONVOYAGE communication system, networking, Information-Centric networking, Internames, Middleware, Namespace, Pub/Sub, API, OpenGeoBase, spatial database.

Table of Contents

List of Figures

List of Tables

Abbreviations

BONVOYAGE Abbreviations	
Abbreviation	**Definition**
API	Application Programming Interface
CCTV	Closed Circuit TeleVision
GPS	Global Positioning System
GTFS	General Transit Feed Specification
HTTP	HyperText Transfer Protocol
ICN	Information-Centric Networking
IoT	Internet of Things
IP	Internet Protocol
IRN	Internames Rendezvous Node
ISL	Internames Service Layer
ITS	Intelligent Transportation System
JSON	JavaScript Object Notation
NPRA	Norwegian Public Roads Administration
NRS	Name Resolution Service
OGB	OpenGeoBase
ORS	Object Resolution Service
TCP	Transmission Control Protocol
URL	Uniform Resource Locator

Table 1: Abbreviations

BONVOYAGE Glossary

Table 2 lists and describes the terms that have been considered relevant in this deliverable.

BONVOYAGE Glossary	
Term	**Definition**
API	A set of routines, protocols, and tools for building software and applications, where a software component is defined in terms of its operations, inputs, outputs, and underlying types and functionalities.
Bonvoyage Communication System	Communication bus available in the BonVoyage platform. It integrates a heterogeneous network architecture, the Internames Service Layer offering standardized and technological-independent networking functionalities, and logical nodes implementing search engine, routing, and rendezvous functionalities. It allows to retrieve contents through request-response and publish-subscribe primitives.
Consumer Proxy	Logical node offering a standardized interface between Data Consumers and the rest of the BONVOYAGE Communication System.
Data Consumer	End user of the BonVoyage platform interested to the information for planning and optimizing transportation services.
Data Producer	Entity of the BonVoyage platform generating and publishing travel-centric information and/or participatory sensing data.
GeoJSON	A format for encoding a variety of geographic data structures.
GTFS	Defines a common format for public transportation schedules and associated geographic information.
Information-Centric Networking (ICN)	Emerging paradigm for the Future Internet, where all information (e.g., a picture) are identified by names that do not include references to its location.
Internames	Federation of heterogeneous networks exposing a ICN interface.
Internames Rendezvous Node (IRN)	Logical node of the BonVoyage Communication System performing publish-subscribe functionalities.

JSON	Syntax for lightweight data-interchange format.
Middleware	Software layer that hides the complexities and the differences of the underlying system's or hardware's technologies, exposing an abstraction layer to the overlying application and allowing the application developer to focus all his/her effort on the task, without the distraction of orthogonal concerns at the system or hardware level.
Internames Service Layer (ISL)	The middleware integrated in the BonVoyage Communication System, implementing Internames functionalities.
Norwegian Public Roads Administration (NPRA)	is a Norwegian government agency responsible for the state and county public roads in the country.
Name Resolution Server (NRS)	Logical node of the BonVoyage Communication System performing routing functionalities.
Named Data Networking (NDN)	Is an ICN oriented project. It is characterized by a hierarchical naming scheme, coupled request and data paths and by-name routing for requests and secure packets with signature.
ONEM2M	Standard for Machine-to-Machine and IoT technologies.
Object Resolution Server (ORS)	Logical node of the BonVoyage Communication System performing search engine functionalities.
OpenGeoBase (OGB)	Federated Spatial Database based on ICN services.
Producer Proxy	Logical node offering a standardized interface between Data Producer and the rest of the BONVOYAGE Communication System.
Soloist	Any software module solving the functionality of ”trip planning”, i.e. providing one or more feasible travel solutions matching specific user requests.

Table 2: BONVOYAGE Dictionary

1 Introduction

1.1 Deliverable rationale

The previous deliverables of WP3 have defined BONVOYAGE ICN network, namely Internames, and its offered communication models: request-response (Deliverable D3.1) and publish-subscribe (Deliverable 3.2).

In this final deliverable of the work package we discuss how it is possible to use these models to support Intelligent Transport System services. Specifically, the considered ITS services are the dissemination of non real-time and real-time ITS information, and the discovery of ITS data sources and planning services (aka soloists).

1.2 Quality review

The internal reviewers responsible of this deliverable are

Version Control Table			
Version n.	**Purpose/Changes**	**Author**	**Date**
v0.1	Section 3	Mauro Losciale [CNIT], Pietro Boccadoro [CNIT], Giuseppe Piro [CNIT], Luigi Alfredo Grieco [CNIT]	12/10/2017
v0.2	Section 2	Giulio Rossi, Michele Orru [CNIT]	15/10/2017
v0.3	Section 2	Andrea Detti [CNIT], Nicola Blefari Melazzi [CNIT]	27/10/2017

Table 3: Version Control Table

1.3 Executive summary

1.3.1 Deliverable description

This Deliverable describes the use of the BONVOYAGE Information Centric Network (ICN), namely Internames, for the **discovery** of ITS data sources and local trip planners (i.e. soloists, see D4.2), and for the **dissemination** of ITS data.

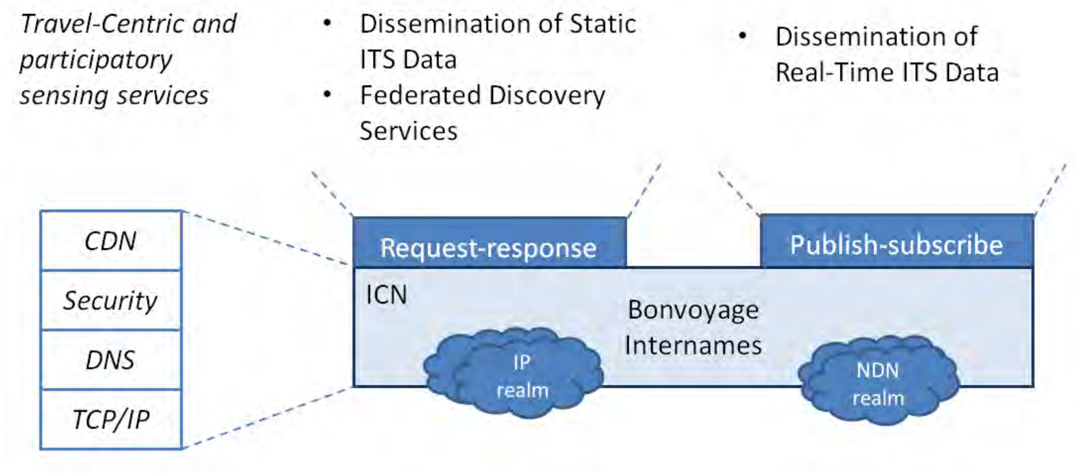

Travel-Centric and participatory sensing services
• Dissemination of Static ITS Data
• Federated Discovery Services
• Dissemination of Real-Time ITS Data
CDN
Security
DNS
TCP/IP
Request-response
Publish-subscribe
ICN
Bonvoyage Internames
IP realm
NDN realm

Figure 1: The BONVOYAGE communications system

As shown in figure 1, Internames and in general ICN paradigm combine in a single network technology several Internet functionality, which are widely used nowadays for the secure dissemination of contents at scale. For instance, DNS name resolution for working with names rather than IP addresses; routing-by-name, caching and multicasting for handling data replication, offloading servers and network links as done by Content Delivery Networks (CDNs); data-centric security for access control and confidentiality, etc.

Internames is actually an *inter-network* whose addressing based on names rather than IP addresses. Several autonomous networks (aka *realms*) can be interconnected each other and it is possible to fetch by name a content published in any realm of the network. The current implementation makes possible to interconnect i) realms based on HTTP/IP technology, e.g. public Internet, private networks, etc.; ii) and native ICN realms, e.g. based on NDN technology.

Internames exports two kinds of name-based communication services : request-response (see Deliverable D3.1) and publish-subscribe (see Deliverable D3.2).

We used request-response API for delivering static ITS data (e.g. NeTEx, GTFS, etc.), similarly to what can be done using HTTP (plus CDN, DNS, etc.). Moreover, we used request-response API to implement a federated spatial database (named OpenGeoBase) that is used to support the discovery of ITS data sources and soloists.

We used publish-subscribe API for the dissemination of geolocalized real-time data. This information can be both generated by end-users/sensors (participatory sensing) and/or by a centralized server (e.g. DATEX II). Authorized publishers inject in the system geolocalized information, while subscribers can register to receive any information related to a given geographical zone (e.g. a box).

In the rest of the deliverable we focus our attention to describe OpenGeoBase as representative ITS application exploiting request-response Internames service, and DATEX II dissemination as representative ITS application exploiting publish-subscribe Internames service. We are not going to discuss the BONVOYAGE use of request-response Internames service for fetching static ITS data (i.e. files) since it is a mere execution of the request-response service described in Deliverable D3.1.

It is worth to remind that practically not all the BONVOYAGE communications get the Internames support, but only the ones for which one-to-many dissemination is expected. Indeed, only in these cases it is worth to exploit efficient dissemination functionality provided by Internames. For instance, a GTFS zip file can be downloaded by many local trip planners (soloists), every time it changes, therefore it is worth to use Internames for its dissemination. A same reason can be done for discovery metadata or DATEX II information. Conversely, ephemeral point-to-point data exchange, such as a orchestrator-to-soloist or user-to-orchestrator SPROUTE requests, are supported by plain HTTP services.

1.3.2 Summary of results

In summary, this deliverable provides the following results:

- Prof of concept about how to use the ICN network of BONVOYAGE to create travel-centric and participatory sensing services
- Design, implementation and performance evaluation of an ICN federated spatial database that can be used to deploy a federated system of National Access Points compliant with the recommendations of the Directive 2010/40/EU
- Design, implementation and performance evaluation of a spatial publish-subscribe service for the efficient delivery of real-time DATEX II information.

2 Federated Discovery Services

An efficient, reliable and secure spatial databases is undeniably a key component of an Intelligent Transport System services. A spatial database can be used both for finding data, e.g. position of a car of car-sharing services, location and availability of the closest parking, etc., or for finding metadata describing where and how a data source or a trip planning service can be accessed, e.g. URL of a local planner/soloist covering part of Norway, or URLs where it is possible to download NeTEx file related to a given region and transport operator.

In BONVOYAGE we devised a federated spatial database, named **OpenGeoBase** [1][2], whose primary goal is to implement a *federated discovery services* which is compliant with the recommendations of the **Directive 2010/40/EU** of the European Parliament and of the Council with regard to the provision of EU-wide multimodal travel information services.

The directive states that [1] :

- *Each Member State shall set up a national access point. The national access point shall constitute a single point of access for users to at least the static travel and traffic data and historic traffic data of different transport modes, including data updates, ..., provided by the transport authorities, transport operators, infrastructure managers or transport on demand service providers within the territory of a given Member State.*

- *National access points shall provide discovery services to users, for example services allowing for the search of the requested data using the contents of the corresponding metadata and displaying such contents.*

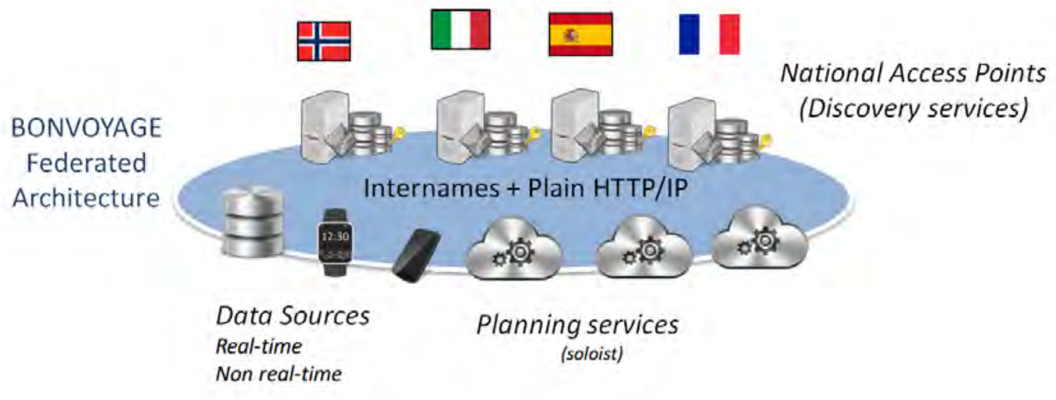


Figure 2: The BONVOYAGE Federated Architecture

[1]https://ec.europa.eu/transport/sites/transport/files/c20173574-multimodaltravelinformationservices-delegatedregulation.pdf

Regarding these recommendations the BONVOYAGE project is exploring a solution that answers to the following technical questions:

- which practical ICT services a NAP should (mandatory) offer ?
- through which API ?
- how can different NAPs interact each other ?

NAP services - Even though ITS data/services and discovery service can be directly offered by the National Access Point (NAP), the BONVOYAGE project focused on a solution where a NAP shall provide only discovery services, which return information (e.g. URLs) making possible the eventual data/service access[2]. The actual data or services can be offered by devices of ITS stakeholders, which transfer the metadata needed for discovery purposes to their NAP (see Deliverable D5.1 for metadata definitions). Such an approach reduces the technical/legal responsibilities (data storage, validity, authentication etc.) and load of a NAP, nevertheless respecting the directive requirements; indeed, through a NAP is possible to discover and thus reach any information (single point of access).

Fig. 2 shows the whole BONVOYAGE federated architecture, as envisioned in Deliverable D1.2. As we can see it is formed by a set of National Access Points, and by data sources and local planning services (soloists) that are provided outside the NAP system, by different ITS stakeholders. The whole system is interconnected either through Internames network or by HTTP services depending on the specific communication service needs, as previously discussed in section 1.

NAP API and interactions with other NAPs - **In the BONVOYAGE vision the whole set of European NAPs forms a federated system exposing the API of a spatial database**, i.e. a federated spatial database. Users can submit to a NAP a discovery request, including also the shape of the interested area (e.g. a geographical box). The NAP will send back the references to all ITS data sources and services matching the searching conditions in the focused area[3]. Each NAP is handled by a different national authority and, furthermore:

- each NAP exposes an API similar the one of a NoSQL spatial database for discovering ITS data source and services. Indeed, spatial API is deemed to be very useful for developing ITS applications. Furthermore, although NoSQL interface may be limiting with respect

[2]Clearly a NAP could also host a data source or service (e.g. DATEX II) but this is not mandatory in our vision

[3]An exemplary OGB based discovery service for GTFS files is available here http://bonvoyage2020.eu/travelcentricservices/

to a SQL one, it strongly simplifies the setup of a distributed deployment like the one we are focusing;

- each NAP locally stores only the (meta) data of ITS data sources and services located in its country;
- each NAP replies to spatial queries/discovery intersecting its country;
- each NAP forwards spatial queries intersecting other countries towards all other NAPs that can have related data, aggregate answers and sends back results to user. This implies that a software developer should contact only a single NAP to access all European data.
- each NAP authenticates its local users for DB operations.

More simply, **a NAP is responsible for handling the discovery of ITS data sources/services located in its country; but through any NAP it is possible to discover all European ITS data source/services.**

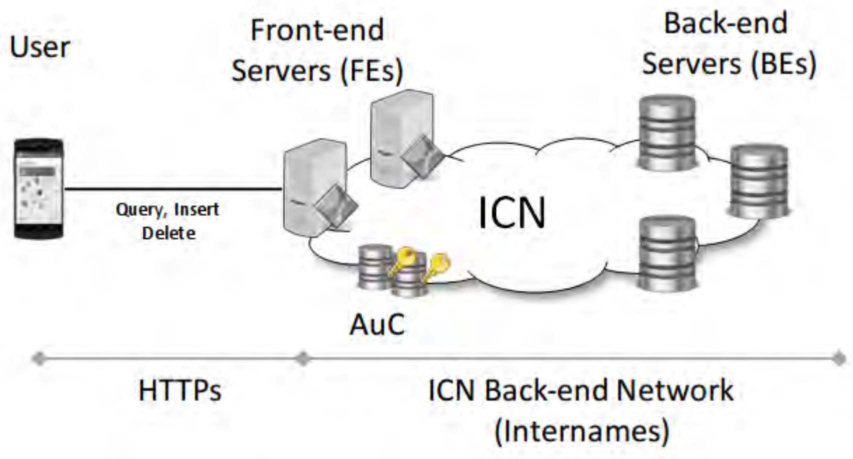


Figure 3: OGB database architecture

OpenGeoBase (OGB) is the federated NoSQL spatial database developed by the project to implement the NAP federated system. OGB is formed by front-end and back-end servers, as shown in fig. 3. We explain technical details of OGB in the remaining sections, but now it is worth to discuss its technical relationship with the NAP federated system and its specific features that we did not find in other distributed databases and that led us to develop a new one. Such features (Table 4) could represent additional technical recommendations for a possible eventual implementation of a European NAP network that we did not find in the directive and that our implementation demonstrates to be feasible.

Using OGB it turns out that a NAP should be composed by:

1. an instance of an OGB front-end server which exposes an NoSQL HTTP API to external users and inter-works with any back-end servers of the federated platform through the services of an Information Centric Network layer (e.g. Internames, NDN, cICN, etc.). Moreover, the front-end inter-works with a local authentication server (AuC) for the authentication of NAP users.

2. an instance of an OGB back-end server storing metadata of data/services located in the NAP country (or of an aggregation of countries) and inter-working with any other front-end server through ICN services. Metadata has actually a GeoJSON format, thus enabling spatial operations.

3. an instance of an authentication server (AuC) interacting with its front-end server for authentication purposes.

Table 4 compares OGB with other existing NoSQL databases highlighting which characteristics make OGB more suitable for implementing a NAP federated system compliant with the Directive 2010/40/EU (and more).

In the following subsections we technically describe the OpenGeoBase architecture, its internal structures and functionalities. Furthermore, we report a performance evaluation, analyzing the behavior of the system with different workloads and operations. We remind that some other OGB related activity are reported in other deliverables. Specifically, another performance evaluation of OGB in case of a realistic load is reported in Deliverable D6.3. The GeoJSON metadata inserted in OGB for the discovery of GTFS, NeTEx and Datex sources and of soloists are reported in Deliverable D5.1. And finally, the security aspect of OGB are discussed in Deliverable 7.1.

2.1 Spatial Database and NoSQL related works

Spatial databases are storage systems specialized in managing data items related to *space* [3]. The space of interest may be real or virtual geographical spaces and spatial data represent information about the location and shape of contained geometric objects. These objects can be points, multi-points, polygons, lines, etc..

Spatial databases provide spatial query capabilities, e.g. proximity queries that return objects close to a point, polygon queries that return objects within polygons, etc. Spatial databases are used for many applications, including Geographic Information System (GIS), navigation software, journey planners, etc.

Spatial databases are usually implemented as extension/plug-in of a general purpose database

Table 4: OGB vs Existing NoSQL for NAP system implementation

	OGB	**Existing NoSQL DB (e.g. MongoDB)**
Data distribution (sharding)	**Geographical** - data are assigned to back-end server on a geographical base. This enables a NAP to handle only (and all) data items located in its reference country	**Hash** - data are assigned to back-end server on the base of an hash function. Thus each NAP could handle data of any other NAP (not in line with EU recommendations).
Query dispatching	**Routing** - Query sent only to those back-end servers that could have data. Consequently, the back-end server within a NAP of a country is only involved for solving discovery requests whose areas intersect the country borders	**Flooding** - Query sent to all back-end servers (useless overload of all NAPs)
Security	**Data-level access control** - multiple users can access the data of a same database table (e.g. the one used for storing ITS discovery metadata), nevertheless user rights depend on the data ownership. Specifically a user can create/update/delete/read its own table entries and only read entries created by other users. No inter-user interference. Possible errors of a user affect only its own data and not data of other users. E.g. a transport operator updating its metadata can not wrongly cancel metadata of another (competitive) transport operator. **Data-level data authentication** - Each DB entry is digitally signed by the data owner (transport operator). Every consumer can verify that each received data is authentic (i.e. equal to the one provided by the transport operator) and thus can not charge NAP authority to provide not-valid information (this problem also motivates us to put data sources outside the NAP)	**Table-level access control** - A user has a right on a whole table. Possible inter-user interference. E.g. a transport operator can unintentionally cancel or modify the discovery metadata inserted by another transport operator thus hampering the proper discovery. **Connection-level data authentication (e.g. only HTTPs)** - Bits transmitted on the network connection are secured by the database server (NAP server), thus DB administrator (NAP authority) is responsible for data authenticity rather than the data owner
Deployment	**Federated** - Many administrators	**Distributed** - Single administrator

management system (DBMS). They may be based either on a relational or on a NoSQL DBMS model. PostGIS is the spatial extension of the popular open-source PostgreSQL relational DBMS, which adds support for geographic objects, allowing spatial SQL queries using several geometries. Being a SQL database it is hard to distribute PostGIS on different servers and performance problems show up when the database size increases. Other (not open-source) SQL databases with spatial extension are Oracle SQL and Microsoft SQL.

For large data-set (Big Data) NoSQL databases are more and more replacing relational ones, since they can be easily distributed over different servers. Indeed, NoSQL databases store generic objects such as JSON ones and each object contains all related data, strongly simplifying the operations to achieve data consistency in distributed environments.

When load increases, the resources of a NoSQL database can be scaled "horizontally" by adding new servers, thus forming a database cluster that is exposed to client applications as a single entity.

Data distribution among servers is carried out by grouping the data by a "sharding-key" and using such key to partition the data set among the (back-end) servers.

Clients interact with one or more front-end servers, which in turn contact back-end servers that satisfy front-end requests exploiting local storage spaces. Typical procedures carried out by front-end servers are: routing of NoSQL operations (queries, insertions, deletions) toward back-end servers; access control; aggregation and post processing of results. The overall architecture may include other kind of servers to perform specific tasks, such as system configuration, security operations, global indexing etc.

MongoDB [4], BigTable (by Google), Cassandra (by Facebook), CouchDB (by Apache) are a popular NoSQL databases with spatial support. With respect to a relational DBMSs, a NoSQL database has fewer instruments for managing query geometry. However, for applications requiring many simple read/write operations on huge data sets, NoSQL databases are deemed to perform better than relational ones since can be easily distributed.

Literature on the use of an ICN network, such as BONVOYAGE Internames, for spatial services is very limited, does not concern database applications, and is mainly focused on vehicular services. In [5] authors propose to label geographical areas with names and use a routing-by-name schemes to carry out location based forwarding in VANET environment. We label geographical areas too, but with the different goal of routing spatial queries. In [6] authors propose to use ICN to dispatch queries towards nodes of a V2X network. The Interest messages (i.e. BV-Request services) contain query conditions in the name; many nodes (vehicles, road side units, etc.) may receive an Interest message (flooding) but only nodes that have data satisfying the conditions will send back an answer. We use ICN for query dispatching too, but do not embed conditions inside an Interest to increase cache hit probability and speedup database server processing; we route Interest only towards the database server that can actually serve the query (no flooding), avoiding useless query processing on other databases, thus limiting system load and latency.

2.2 OpenGeoBase

2.2.1 Offered services

OpenGeoBase (OGB) is a NoSQL spatial-database where users can stores geo-referenced information, structured as GeoJSON objects [7].

For instance, a geo-referenced parking application could use the following GeoJSON format to represent the presence of a parking area in GPS coordinate 12.4873258E, 41.8704346N.

```
 {
"oid":"012/041/parking/bv/parkingrome/123"
"type":  "Feature",
"geometry":  {"type":  "Point","coordinates":  [12.4873258, 41.8704346]},
"properties":  {"park-name":  "Parkbus"}
}
```

Users can carry out either inclusion or intersect range-queries for obtaining all the GeoJSON objects which are completely (inclusion) or partially (intersect) contained in a range-query shape, i.e. a 2D bounding box or a polygon.

User can interact with OGB through a HTTP Rest API. Furtermore, current implementation provide also a JAVA SDK (see Deliverable 6.1).

As previously said, BONVOYAGE uses OGB to create a federated system storing metadata that describes ITS data sources and trip planning services (soloist). The metadata has the GeoJSON structure described in Deliverable 5.1. We have defined, for instance, GeoJSON metadata for GTFS, NeTEx, DATEX II, soloists, etc. The setup of GeoJSON metadata from actual information sources is carried out by libraries forming the so called Metadata Handling Tools.

2.2.2 Architecture

Figure 3 depicts the OpenGeoBase architecture. It is formed by a distributed cluster of servers working on top of an ICN layer (such as Internames), rather than on a TCP/IP one. The front-end servers are ICN consumers, and the back-end servers are ICN producers. Applications interact with front-end servers that expose a set of API for querying, inserting and deleting objects. Application can connect to the Front-end server via HTTPs.

In what follows we describe OGB functionality and procedures assuming that it is entirely deployed on a NDN Internames realm, i.e. the ICN services interconnecting front-end and back-end servers are offered by a NDN network. In this case we remind that Interest messages are used to request data items sent back within Data message [8]. Each network nodes has a cache of Data messages and forwards Interest messages by using a name-based longest prefix matching, which uses a a name-based forwaring information base (FIB). FIB entries are configured by a name-based routing protocol, NLSR in our case [9].

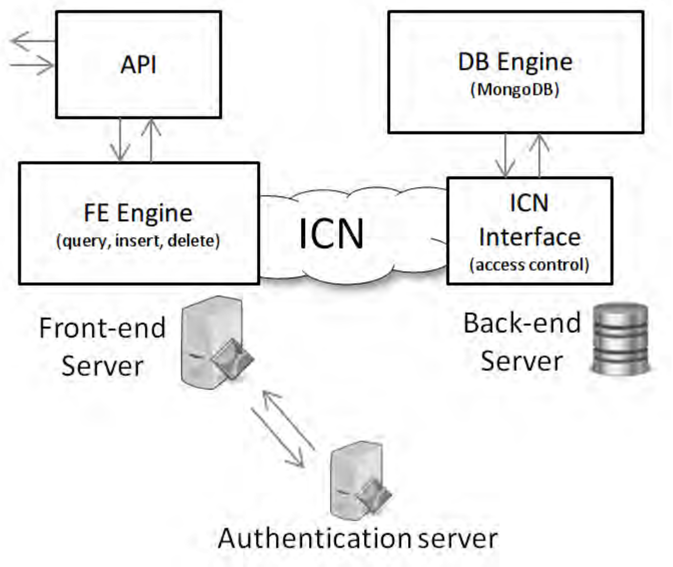


Figure 4: Front-end and back-end server functionality

Figure 4 shows the *functional* decomposition of the OpenGeoBase architecture:

- Front-end server: offers an Application Programming Interface (API) and an internal engine that satisfies client requests by using the ICN based procedures. The front-end server carries out the query dispatching, identifying what are the back-end servers that contain the required information, or that should contain the information that the user wants to enter in the system in case of an insert procedure. For each back-end server an Interest message will be sent and routed by ICN towards the proper back-end server.The Front-end collects response data and sends them back to the client. The Front-end servers can be developed considering the ability to work in fat client or thin client architectures. In fat client architectures the front-end server run in the same machine of the application so they can communicate within a local interface. In the thin client case the front-end server and the application run in two different machines, so they can communicate through the HTTP or HTTPS protocol.

- Back-end server: are composed by an ICN interface that deals with ICN packets received-from or going-to the front-end servers. When the back-end server receives an ICN packet it extracts the information from the Interest name and (in case of an insertion) from the Data content and contact the local DB engine to store, retrieve or delete the data. When the local database engine returns the requested data the back-end sends it to the Front-end within an ICN Data message (or more than one Data message in case of the response exceeds the ICN packet size)
- Local database engine: handles the local storage space and, in current implementation, it works above a MongoDB driver, indeed we are using MongoDB as local database. The local engine performs queries and insertions on the local database and is responsible for indexing and logical separation of data (for each tenant a database is reserved and for each tenant's dataset a collection is created)
- Authentication server: if OGB is implemented using a thin client approach, the system has a dedicated repository for certificates and users private/public keys, namely Authentication (AuC) server. When an user performs an authentication on OpenGeoBase, front-end fetches from AuC users permissions and other personal informations and stores them temporarily in order to perform further actions (e.g. queries or insertions of data, if allowed). In thin client architecture, Data signature is performed by front-end using users identity. Else, in case of a fat client approach, AuC is anyway present but it stores only certificates. Users private/public keys is stored in client application directly and signature is performed by user directly. Security and privacy functionalities are widely described in D7.1.

The system administrator(s) can deploy a distributed set of front-end servers , back-end servers and AuC servers. Each back-end server can be dedicated to store data related to a zone of the world, e.g. a country. Different front-end and AuC servers instead can be deployed for load balancing or other purposes. The distributed set of servers can have a unique administrator (distributed deployment) or multiple administrators (federated deployment).

2.2.3 Storage model

The OpenGeoBase platform is multi-tenant, where a tenant is a real (user) or virtual entity that gets a slice of storing space. The storing space of a tenant could be organized in *dataset*, to divide and logically classify data. A tenant/slice has users that access OGB data to read or write GeoJSON objects (fig. 5).

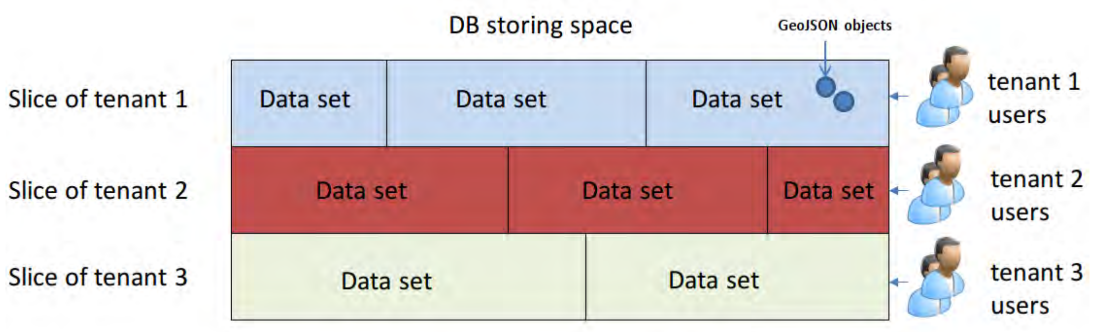


Figure 5: OGB storage model

OGB read and write access operations are secured through a data-centric security approach. Users can read and write data only within the slice of their tenant. A user of a tenant can access (read) the data contained in the slice, and can or cannot carry out insert/delete/modify operations. In any case delete or modify operations can be carried out only on the data items of the user and not on items inserted by other users (data-level access control).

Users reading data through queries are sure about the ownership and the integrity of the returned GeoJSON items exploiting ICN data-centric security, for which each item is signed by the owner and each item also includes the owner identity.

OGB users can carry out either inclusion or intersect spatial range queries for obtaining all the GeoJSON objects completely (inclusion) or partially (intersect) contained in the range area, e.g. a 2D bounding box or a polygon.

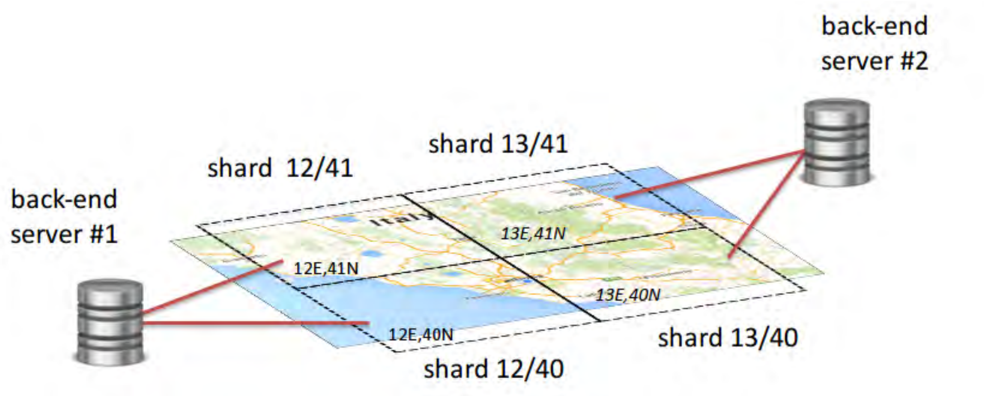


Figure 6: Data sharding

2.2.4 Data sharding

Sharding is a method for distributing database objects across multiple back-end servers. It uses an object attribute (*sharding key*) and a *sharding logic* to map the object onto a *sharding* *domain*, which is partitioned in subsets called *shards*. Each back-end server is configured for storing the objects belonging to one or more shards.

To ensure optimal performance and scalability the sharding strategy has to be selected in a way that is appropriate for the types of queries the application performs. When the sharding strategy is carefully designed, most of the queries can be sent only to those back-end servers that actually have the interested data (query routing), optimizing system load. Otherwise, the queries should be distributed to all the back-end servers (query flooding).

Accordingly, OGB uses a geographical sharding strategy, being spatial queries the most expected ones for ITS discovery services. The sharding domain is the physical space. Each shard is a square geographical tile, aligned with the lng/lat GPS grid, and whose side is 1 degree long. The sharding key is the geometry field of the GeoJSON entry.

To support data sharding through ICN means, each shard has a unique *shard identifier* (sid) equal to `lng/lat`, where lng/lat is the south-west coordinate of the geographical tile the shard refers to. For instance, in fig. 6 we have two back-end servers and each server handles two different shards. Related shard identifier are 012/041, 013/041,012/040,013/040.

On the ICN routing plane, back-end servers advertise their shard identifiers as ICN name prefixes. Thus, ICN nodes can route-by-name Interest packets having the shard identifier as first name component towards the proper back-end server.

2.2.5 ICN data structures

OpenGeoBase uses two kinds of ICN Data structures, namely *oData* and *qData*. Both are ICN Data messages but whose content is different in the OGB framework.

An oData (object-Data) contains a GeoJSON object and its name (*oName*) is obtained by combining the shard identifier (*sid*) of the geographical tile containing the object, the data set identifier (*did*), the tenant-identifier (*tid*), the unique user identifier (*uid*) and a unique application-dependent suffix (e.g., a random string), as follows:

```
oName = {sid}/{did}/{tid}/{uid}/{app suffix}
```

where the `did` identifies the logical data set to which the oData belongs (e.g. parking), the couple `tid/uid` identifies an unique user of the OGB platform and it's formed by the tenant identifier ('tid') and the user identifier ('uid') of the GeoJSON (e.g. bv/parkingrome). Finally, the `app suffix` is an application suffix making the name unique, e.g. a random number (123).

A qData represents the result of a query sent to the OGB platform, and contains the list of the oNames whose objects match the query conditions.

2.2.6 Local database engine

Each OGB back-end server uses a MongoDB as local databases to store oData and other related information. Actually, two MongoDB databases are deployed for each tenant.

For each GeoJSON object contained in a oData, the first database stores the couple ⟨ geometry, oName ⟩ extracted by the oData. This database uses the MongoDB *2dsphere* spatial indexing on the base of the geometry field and is used to answer spatial queries (qInterest), as explained in the following section.

The second database stores the couple ⟨ oName, oData ⟩, a plain MongoDB indexing on oName is used, and it is used to answer object queries (oInterest), as explained in the following section.

Each tenant can create multiple data set: for each of them the back-end servers create a MongoDB collection in the tenant databases, to get a logical separation of the data and a quicker search.

It is worth to note that other local database engines different from MongoDB can be used. For instance in a former implementation we used SQLite3, but it did not provide spatial indexing features and thus we moved to MongoDB for decreasing response time.

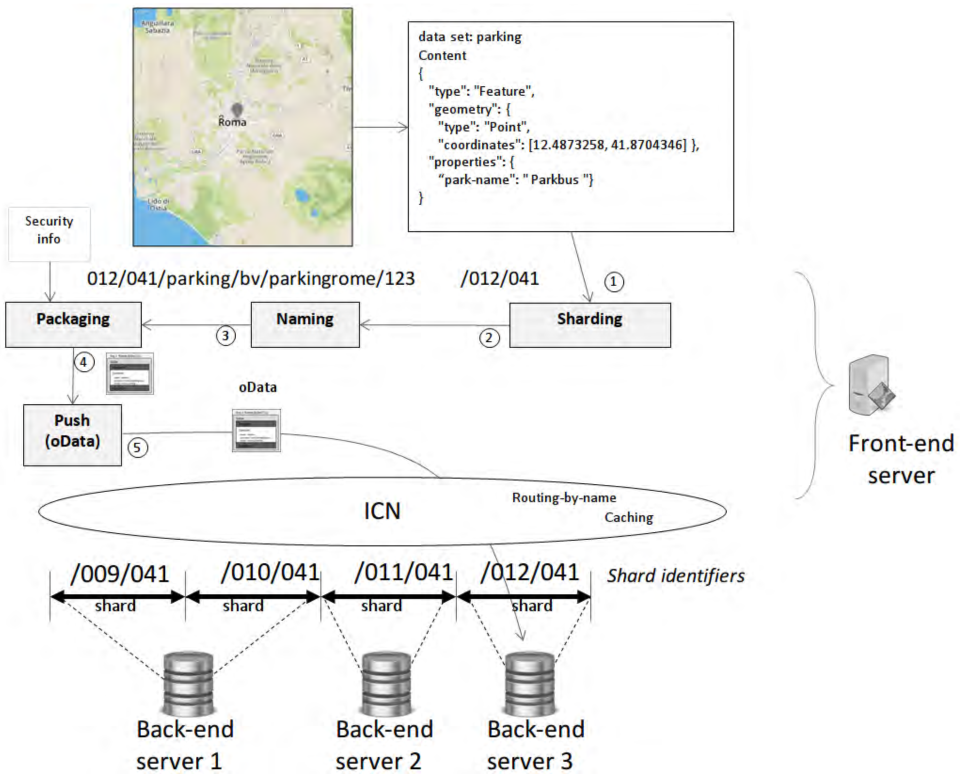
2.2.7 Procedures

Insert Operation From the top down, fig. 7 describes the object insert procedure of the OGB database. When a user inserts an object, the sharding logic computes the associated shard identifier on the base of the geographical information of the object.

A naming function composes the object identifier, i.e. an oName
(e.g. 012/041/parking/bv/parkingrome/123). Afterwards, a packaging function encodes the object as an oData packet and, finally, a Push procedure exploits ICN routing-by-name to deliver the oData packet to the responsible back-end server advertising the related shard identifier, which stores it in its local database.

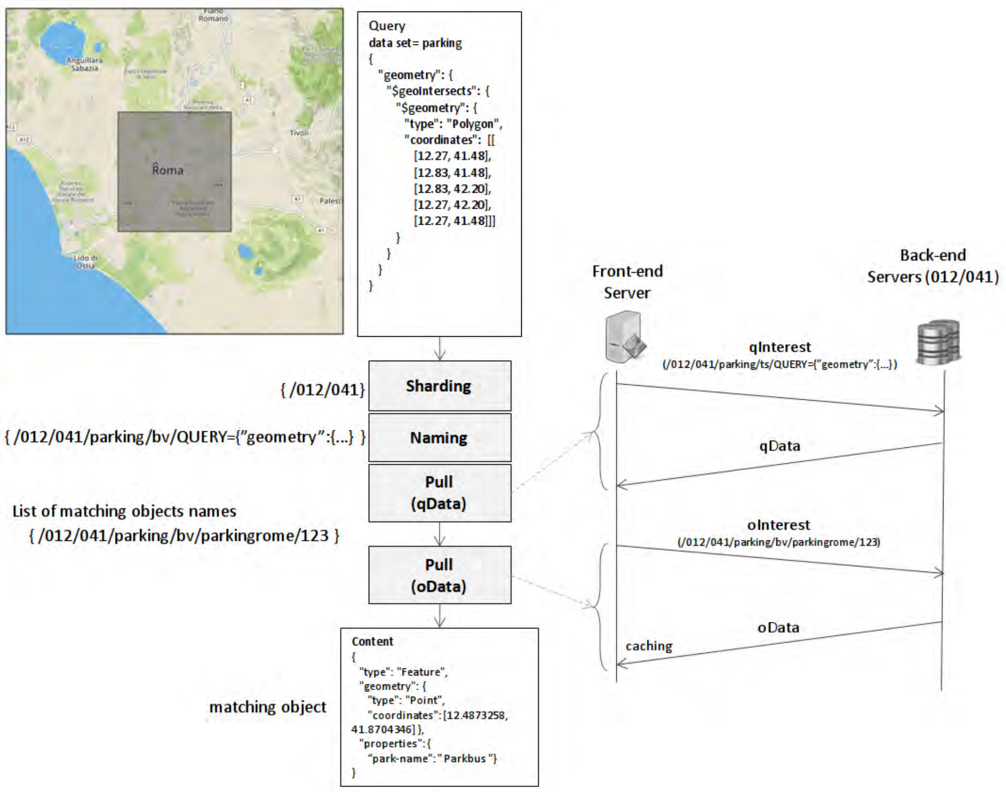
For some applications, it may happen that an object ”intersects” more than one shard (e.g, a polygon GeoJSON representing a large portion of space). We deal with a multiple shards object by packaging it in many oData packets, one for each shard, whose oNames differ for the shard identifier. We identify one of them as *master* object, and the others as *reference* objects. The reference objects contain the name of the master object, i.e., a reference.

Query operation A query is a request for database objects satisfying specific conditions. A query is solved by the front-end server as shown in fig. 8.



```
data set: parking
Content
{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [12.4873258, 41.8704346] },
  "properties": {
    "park-name": " Parkbus "}
}
```

Figure 7: Data insert


Roma
{ /012/041}
Sharding
{ /012/041/parking/bv/QUERY={"geometry":{...} }
Naming
Pull (qData)
List of matching objects names
{ /012/041/parking/bv/parkingrome/123 }
Pull (oData)
matching object
Front-end Server
Back-end Servers (012/041)
qInterest
(/012/041/parking/ts/QUERY={"geometry":{...})
qData
oInterest
(/012/041/parking/bv/parkingrome/123)
oData
caching

```
Query
data set= parking
{
  "geometry": {
    "$geoIntersects": {
      "$geometry": {
        "type": "Polygon",
        "coordinates": [[
          [12.27, 41.48],
          [12.83, 41.48],
          [12.83, 42.20],
          [12.27, 42.20],
          [12.27, 41.48]]]
      }
    }
  }
}
```

```
Content
{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates":[12.4873258,
41.8704346] },
  "properties":{
    "park-name":" Parkbus "}
}
```

Figure 8: Generic query procedure

The sharding logic parses query spatial parameters and computes the involved shards (geographical tiles). For each shard, a naming function computes a name, called *qName*, which is composed by the shard identifier (sid), the data set identifier (did), the tenant identifier (tid) and query conditions, e.g. the coordinates of the requested portion of space represented as JSON object in order to be correctly parsed from MongoDB at back-end side.

```
qName = {sid}/{did}/{tid}/{query conditions}
```

For each qName an Interest, called *qInterest* (query Interest), is sent out and it is routed-by-name by the ICN towards the appropriate back-end server on the base of the shard identifier. The receiving back-end server parses the qName to extract the query conditions, and then carries out a local query using the local MongoDB driver. The local query returns only the names (oNames) of the master objects that match the query conditions. Such a name list is packaged in a Data packet, called *qData* (query Data), which is sent back to satisfy the qInterest.

When all qData packets are received from the possibly multiple back-end servers, the front-end server has a whole list oNames, which are then pulled through an oInterest-oData (object Interest - object Data) packet exchange. In so doing, we are actually solving a query in two *pull* phases and this may sound as a temporal inefficiency. Anyway, we have chosen this approach both to transport only one time the objects that intersect more shards, and to exploit the ICN in-network caching as hereafter discussed.

Furthermore, is it possible to reduce the number of qInterests, if more than one shard obtained from the sharding function is managed by the same back-end server. In this case, only one qInterest is created for each involved back-end server.

Finally, OGB support also non-spatial queries, i.e. query not including geographical information. In this case, sharding function will return a qName for each back-end server present in the system, in so doing the query is actually broadcasted.

Caching Even though caching can dramatically accelerate query processing, its usage should be carefully designed in database applications, where it is likely not acceptable to send back stale data. For this reason, we use two different caching strategies for qData and oData.

We observe that while the name of a qData may remain unchanged (e.g. the same query repeated at different times), its content (a list of matching object names) may change over time due to object insertions or removals. As a consequence, we do not cache qData packets within the ICN forwarders, since they use an expiry-based freshness control, and therefore stale data may be sent back.

For what concerns the caching of oData packets, the cache inside the ICN forwarding engine can be safely used, since their content never change. Moreover, when an object is removed, its oName will no more be included in any qData, thus the removed oData will be never fetched during the query procedure, even if cached.

Delete operation To delete an object, it is necessary to remove the master object and, in case, all of its reference objects. The removal of an object is carried out by issuing a command Interest, called *dInterest*, which includes the oName of the object to be removed, followed by the "/DELETE" command string. A Data message called *dData* containing the operation result is sent back. Usually the user reminds only the name of the master object. In this case the deletion procedure is carried out firstly pulling the master object; then computing the involved shards and names as during the insert phase, and finally removing e one by one the master and the reference objects using dInterest packets.

2.3 Performance Evaluation

We carried out a performance evaluation using two European real data sets, relevant to Intelligent Transport System (ITS) applications. The first data set, named "GTFS", contains information regarding European public transports such as: stop coordinates, schedules, timetables, etc. This information has been derived by downloading a thousand public GTFS files from Internet. According to Deliverable D5.1, for each GTFS file, we inserted a GeoJSON multipoint object where each point is associated to a stop. Several stored objects are composed by thousands of points (e.g. all train stops of a country) and their size may even reach the MByte order.

The second data set, named "Rome bus", is spatio-temporal data set and contains the positions of 2870 buses in Rome, sampled with a step of 5 minutes during a single day. Each bus measurement is stored as a GeoJSON point object, and includes bus detail, position and sample time.

The considered cluster architecture is formed by a set of front-end (FE) servers, a set of back-end (BE) servers, and a benchmark application that uniformly distributes queries and insertions among available FE servers. These software components run on different virtual machines, connected to each other by a Linux bridge.

Fig. 9 reports the maximum range query rate for the GTFS data set versus the range query area, for different cluster configurations, i.e. number of FE and BE servers. Maximum rate is the highest rate for which the time needed to solve a query (query delay) has a stable behavior versus time, as shown in fig. 10. It has been measured by loading the system with a sequence

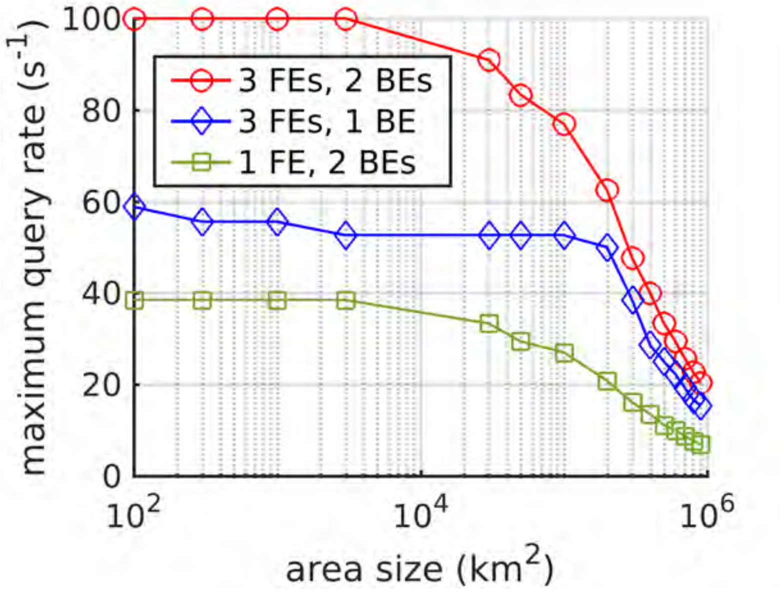


Figure 9: Maximum query rate vs query area, GTFS data set, Poisson inter-arrivals

of range queries, randomly located in Europe, and whose inter-arrival time follows a Poisson distribution.

The cluster with the greatest resources, 3 FE and 2 BE servers, supports the highest rate, thus confirming the database capability to horizontally scale. Performance improves both increasing the number of BE servers, thanks to data sharding and query routing, and increasing the number of FE servers, thanks to load balancing. The maximum rate decreases by increasing the range query area, since queries require more processing and data transfer.

Fig. 11 shows the average query delay versus the area size, in case of an unloaded system. Unloading conditions are reproduced by submitting range queries with a rate equal to the half of the maximum one. There is no practical difference among the cluster deployments since increasing resources, i.e. FE or BE servers, is only needed in overloading conditions. We point out that the maximum rates reported in fig. 10 are much greater than the inverse of query delays reported in 11. However, this result is not surprising since it is a consequence of our multi-threads implementation of the FE server, whereby client requests can be served in parallel.

Fig. 12 reports the average query delay versus the query rate in case of range queries of 10^5 km^2. From the comparison between 3FE-1BE and 1FE-2BE, we infer that for our OGB it is more effective to increase the number of FEs than the number of BEs. In facts, the processing load of a FE server is greater than the one of a BE server. It is worth to note that delays of 1FE-2BE deployment rapidly grow up, since we are reaching its maximum sustainable rate.

Fig. 13 shows the maximum query rate in case of spatio/temporal queries carried out on

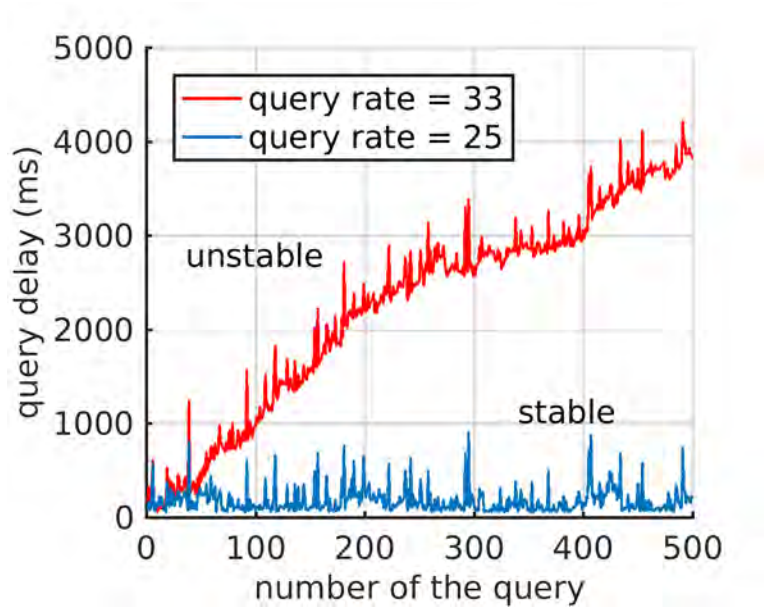


Figure 10: Query delay vs query number, GTFS data set, Poisson inter-arrivals, query area 10^5km^2, 1 FE, 2 BE

the Rome bus data set. We consider range queries whose spatial extension is 1 km^2, randomly located in Rome, and requesting objects that are valid in a given period. Being Rome fully contained in a single shard (12 lat, 42 lng), we used a single back-end server and only changed the number of front-end servers. Also, in this case having more resources makes it possible to sustain a greater rate. Increasing the query temporal extent, query rate decreases since more objects are sent back, and also the front-end decomposition processing is higher.

Fig. 14 reports the maximum insert rate measured by storing spatio/temporal objects with point geometry. Also for insertions, performance improves scaling out the cluster resources. We remark that differently from traditional databases, in our case each query or insertion requires a digital signature verification and/or computation, which adds few milliseconds of processing delay but provides data-level access control feature. To give an idea of the security impact, in fig. 14 we also reported a case in which inserted objects are unsigned.

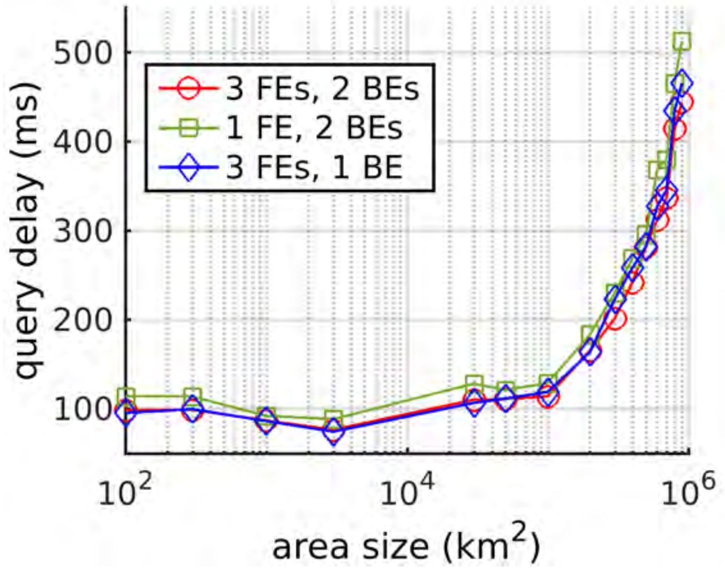


Figure 11: Average query delay vs query area, GTFS data set, Poisson inter-arrivals, query rate equal to the half of maximum rate (unloaded condition)

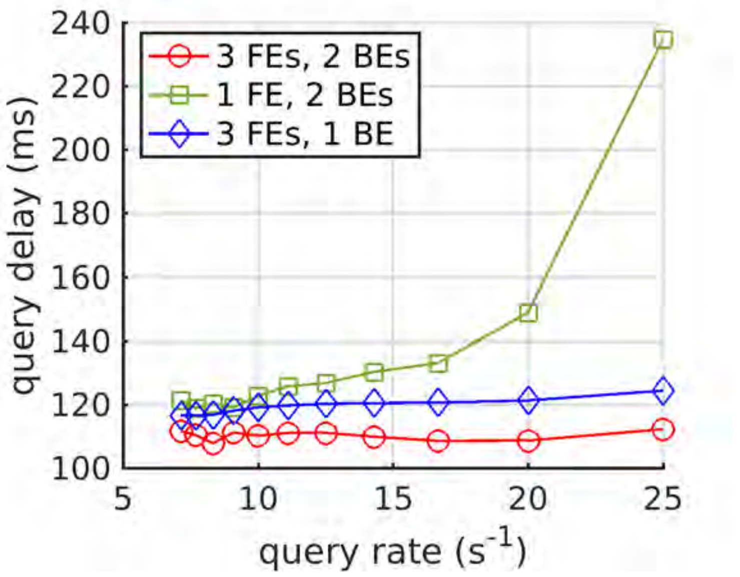


Figure 12: Average query delay vs query rate, GTFS data set, Poisson inter-arrivals, query area 10^5km^2

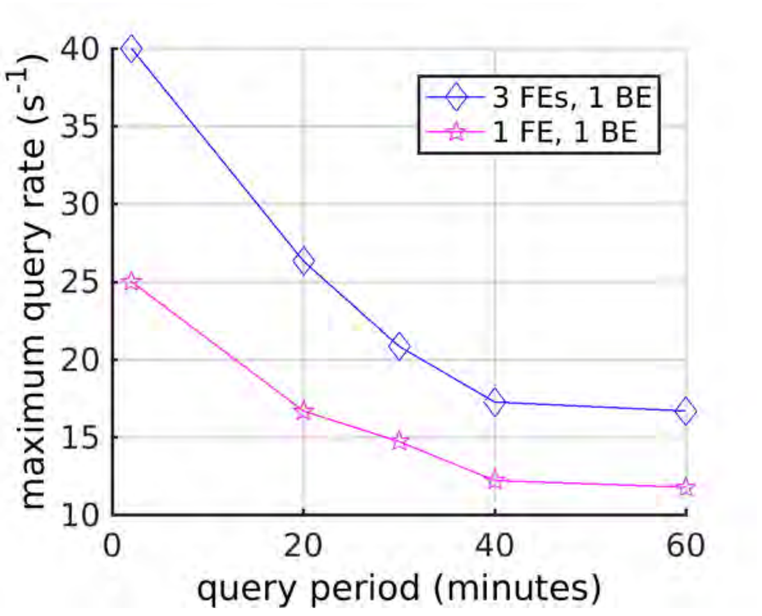


Figure 13: Maximum query rate vs temporal query size, Rome bus data set, Poisson inter-arrivals, query area 1 km^2

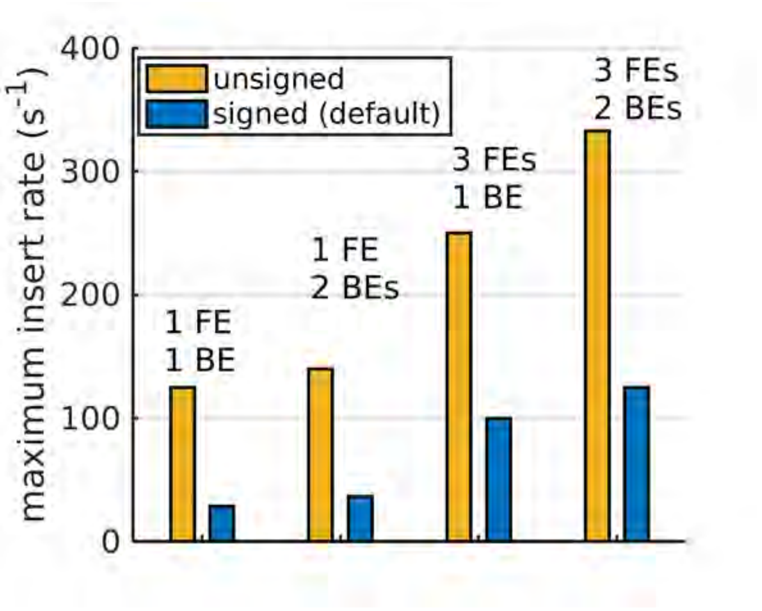


Figure 14: Maximum insert rate vs number of FE/BE servers, objects with point geometry, Poisson inter-arrivals, signed (default) and unsigned objects

Service	URL
Weatherdata	https://www.vegvesen.no/ws/no/vegvesen/veg/trafikkpublikasjon/vaer/2/GetMeasurementWeatherSiteTable https://www.vegvesen.no/ws/no/vegvesen/veg/trafikkpublikasjon/vaer/2/GetMeasuredWeatherData
Web Camera	https://www.vegvesen.no/ws/no/vegvesen/veg/trafikkpublikasjon/kamera/2/GetCCTVSiteTable
Situation	https://www.vegvesen.no/ws/no/vegvesen/veg/trafikkpublikasjon/trafikk/2/GetSituation

Table 5: Services exposed by the NPRA

3 Dissemination of real-time ITS data: the DATEX II use-case

In general, the BONVOYAGE Communication System can be used to disseminate any kind of contents. Data Producers, however, must properly arrange their data, thus making their representation in line with the design principles presented in the previous Section. To give a concrete example, attention is focused on travel-centric data formatted according to the DATEX II standard [10] (http://www.datex2.eu) thus explaining the set of advantages provided by the BONVOYAGE Communication System. Specifically, DATEX II represents one of the outcome of the standardization process related to Delivering European Transport Policy and can be used to expose traffic information through XML files. In [10] DATEX II data model has been used in order to exploit the description of parking lots. Comparing with the parking publication extension, DATEX II model describes some dynamic parking attributes.

The use case considers DATEX II data referring to Norway, exposed by Norwegian Public Roads Administration (NPRA), the national company in charge of planning, building, operating and maintaining national and county roads. NPRA has a web server exposing data related to traffic conditions, information coming from different sensors on the roads (like weather monitoring stations), and Closed Circuit TeleVision, which are summarized in Table 5 with the related URLs available at the date of delivery of the present contribution. A registration for an account on the NPRA website is required for downloading the data.

Currently, a user interested in retrieving contents referring to a specific geographical area must perform the following tasks: (1) he must periodically download all the files from the remote web server, (2) process them, (3) identify updates, if any, and (4) extract only the information of his interest. Unfortunately, these operation are extremely resource consuming. However, considering that the percentage of information update for each XML file is, on average, always less than 10% as shown in Figure 15, these operations result to be even more inefficient. In particular, the percentage varies

The BONVOYAGE Communication System aims at optimizing the dissemination of this kind of data, through a novel communication system based on ICN principles. Fortunately, each DATEX II record has geographical information. Therefore, it is possible to extract travel-

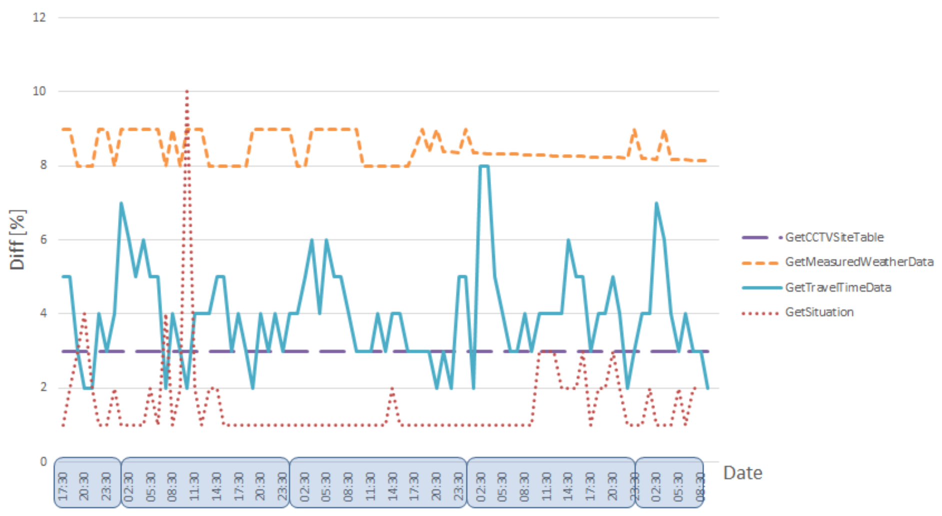


Figure 15: DATEX II update information

centric contents from each single XML file and use them in the BONVOYAGE Communication System as independent data. As a consequence, a customized processing of original DATEX II files offers the possibility to make optimized geo-referenced queries, retrieve a specific selection of DATEX II data exposed by the remote server, as well as collect future updates through an efficient publish-subscribe mechanism.

The BONVOYAGE Communication System is in charge of disseminating travel-centric data across sources and soloists. To optimize the whole dissemination process, original data have to be properly processed and exposed through information-centric and geo-referenced approaches.

Through a DATEX II files, it is possible to expose information about:

- *Road situations*: the information in GetSituation files are structured in a tree fashion using situation tags to mark all the event regarding a single road. Every situation have one or more situation record fulfilling information with the aim of specifying details about the specific accident or maintenance work or similar events happening in that precise road. Every condition is geo-referenced using GPS coordinates and the information are completed with specific comments. Some of the texts are also used for Variable Message Sign (VMS).

- *Weather stations*: on the roads, in precise positions, weather stations are placed. A number of different sensors might be present and for each of them publications are gathered. GetMeasuredWeatherData files collects the publications coming from all the weather stations which are identified by a unique id. Every station has a number of associated sub ids which are related to sensors: for every kind of sensor (e.g.: temperature, wind, etc.) a specific sub id is associated. This implies that not all the stations will publish for every kind of possible data.

- *Web Camera*: camera dedicated files are giving information about every single camera installed on the roads. The information is given thanks to a unique identification number through which it is possible to understand where is located, with a specific reference to the road, its type and the county. It is also specified which kind of model the camera is. The picture taken in a specific moment is also documented. The latter is stored and published on a website and the address to which the information can be consulted is one of the information within the tree. Furthermore, it is specified in which format the picture is stored, together with the size it has.

3.1 Namespace

The BONVOYAGE Communication System adopts a realm-based, geo-referenced, and hierarchical namespace, as described in Figure 16.

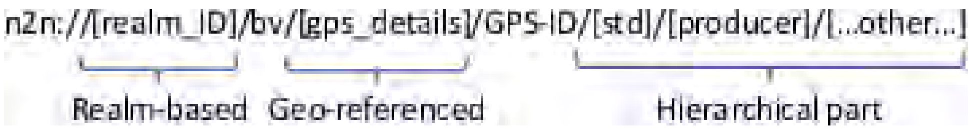
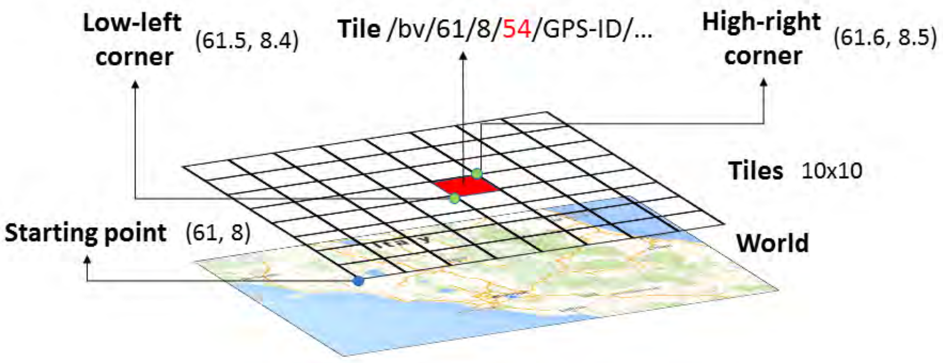

n2n://[realm_ID]/bv/[gps_details]/GPS-ID/[std]/[producer]/[...other...]
Realm-based
Geo-referenced
Hierarchical part

Figure 16: Namespace structure

First of all, the name starts with a string identifying the realm where the Data Producer is located (i.e., [*realm_ID*]). This information is used by Internames for properly routing consumers' requests within the heterogeneous network infrastructure. In line with the OpenGeoBase [1] approach, travel-centric contents can be fetched according to their geographical information. This implies a spatial indexing of available contents through a layered grid, aligned with world's parallels and meridians. Grid regions are called *tiles* and present a geographical extension of $100km^2$. Therefore, the two Global Positioning System (GPS) points provided in [*gps_details*] simply identify a specific tile of a layered grid, where the content belongs to. Finally, the remaining part of the namespace is used to carry information about the standard adopted to expose the considered travel-centric data (i.e., [*std*]), the Data Producer (i.e., [*producer*]), and any other characteristic details (i.e., [*..other...*]). For instance, a content available in a geographical area having a tile described with GPS coordinates 8.4 and 61.5, is identified as shown in Figure 17.

The proposed approach assumes to divide the entire Norway in tiles of $100km^2$ each. For every tile, 4 different travel-centric contents are identified, which refer to the different XML files



Low-left corner (61.5, 8.4)
Tile /bv/61/8/54/GPS-ID/...
High-right corner (61.6, 8.5)
Tiles 10x10
Starting point (61, 8)
World

Figure 17: Example of spatial indexing

exposed by the NPRA web server. This brings to an overall dataset with 43600 content names. In Figure 18 are depicted the available items in the dataset, mapped with the geographical coordinates.

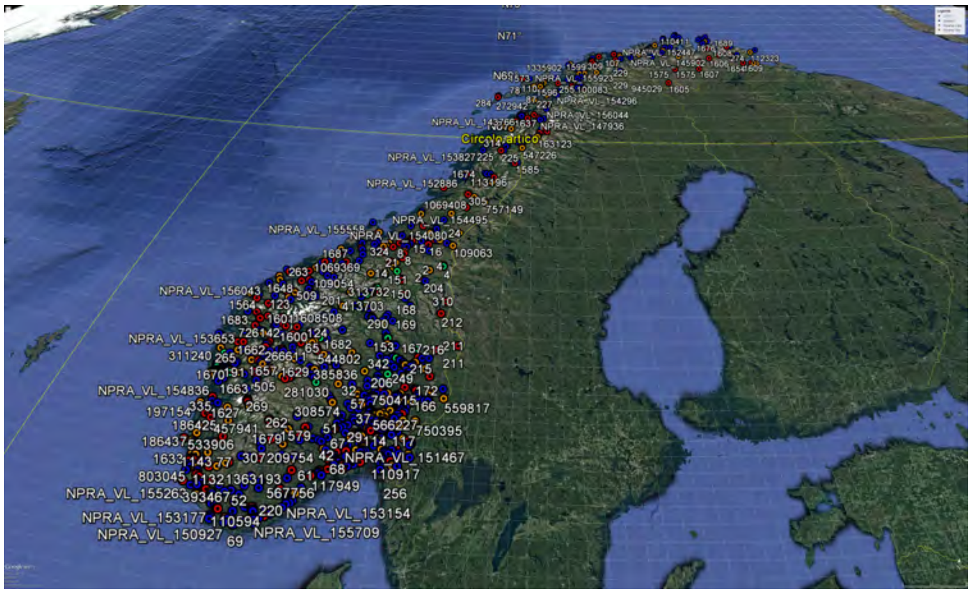


Figure 18: Distribution of the NPRA contents

A unique name is defined for each tile and for each type of content. Therefore, by focusing the on a tile whose low-left corner has GPS coordinates equal to (latitude = 59.9, longitude = 10.7), the names associated to corresponding contents are shown below:

```
n2n://polibaNet/bv/10/59/79/GPS-ID/datexii/npra/situation/init
n2n://polibaNet/bv/10/59/79/GPS-ID/datexii/npra/wheatherdata/init
n2n://polibaNet/bv/10/59/79/GPS-ID/datexii/npra/wheatersite/init
n2n://polibaNet/bv/10/59/79/GPS-ID/datexii/npra/cctv/init
---------------------------------------------------------------------
```

```
n2n://polibaNet/bv/10/59/79/GPS-ID/datexii/npra/situation/update
n2n://polibaNet/bv/10/59/79/GPS-ID/datexii/npra/wheatherdata/
   update
n2n://polibaNet/bv/10/59/79/GPS-ID/datexii/npra/wheatersite/
   update
n2n://polibaNet/bv/10/59/79/GPS-ID/datexii/npra/cctv/update
```

In the following are reported for each service an example of records generated from the Data Producer in the reference tile.

Listing 1: Closed Circuit TeleVision (CCTV) extracted item

```
<d2LogicalModel xmlns="http://datex2.eu/schema/2/2_0"
   modelBaseVersion="2">
  <exchange>
    <supplierIdentification>
      <country>no</country>
      <nationalIdentifier>Norwegian Public Roads Administration</
         nationalIdentifier>
    </supplierIdentification>
  </exchange>
  <payloadPublication xmlns:xsi="http://www.w3.org/2001/XMLSchema
     -instance" lang="nob" xsi:type="GenericPublication">
    <publicationTime>2017-10-12T13:34:12.218+02:00</
       publicationTime>
    <publicationCreator>
      <country>no</country>
      <nationalIdentifier>Norwegian Public Roads Administration</
         nationalIdentifier>
    </publicationCreator>
    <genericPublicationName>CctvSiteTablePublication</
       genericPublicationName>
    <genericPublicationExtension>
      <cctvSiteTablePublication>
        <headerInformation>
          <confidentiality>noRestriction</confidentiality>
          <informationStatus>real</informationStatus>
        </headerInformation>
        <cctvCameraList id="1" version="1">
          <cctvCameraListVersionTime>2017-10-12T13:34:12.218+02
             :00</cctvCameraListVersionTime>
          <cctvCameraMetadataRecord id="297634" version="1">
            <cctvCameraIdentification>0329002</
               cctvCameraIdentification>
            <cctvCameraRecordVersionTime>2017-10-12T13:34:12
               .190+02:00</cctvCameraRecordVersionTime>
            <cctvCameraSiteLocalDescription>
              <values>
                <value lang="nob">Aker sykehus</value>
              </values>
            </cctvCameraSiteLocalDescription>
            <cctvCameraOrientationDescription>Retning  s  r</
               cctvCameraOrientationDescription>
            <cctvCameraType>colourCcd</cctvCameraType>
            <cctvCameraLocation xsi:type="Point">
              <locationExtension>
                <locationExtension>
                  <roadNumber>R4</roadNumber>
                  <countyNumber>3</countyNumber>
                </locationExtension>
              </locationExtension>
              <pointByCoordinates>
                <pointCoordinates>
                  <latitude>59.941025</latitude>
                  <longitude>10.798009</longitude>
                </pointCoordinates>
              </pointByCoordinates>
            </cctvCameraLocation>
            <cctvStillImageService>
```

```
            <cctvStillImageServiceLevel>1</
               cctvStillImageServiceLevel>
            <cctvStillImageFormat>jpeg</cctvStillImageFormat>
            <stillImageUrl>
              <urlLinkAddress>http://webkamera.vegvesen.no/
                 kamera?id=297634</urlLinkAddress>
              <urlLinkDescription>
                <values>
                  <value lang="nob">http://www.yr.no/sted/Norge
                     /Oslo/Oslo/Aker_sykehus/</value>
                </values>
              </urlLinkDescription>
            </stillImageUrl>
          </cctvStillImageService>
          <cctvVideoService>
            <cctvVideoServiceLevel>1</cctvVideoServiceLevel>
            <cctvVideoEncodingStandard>hls</
               cctvVideoEncodingStandard>
            <cctvVideoFrameRate>1</cctvVideoFrameRate>
            <videoUrl>
              <urlLinkAddress>https://kamera.vegvesen.no/public
                 /0329002_1/manifest.m3u8</urlLinkAddress>
              <urlLinkDescription>
                <values>
                  <value lang="nob"/>
                </values>
              </urlLinkDescription>
            </videoUrl>
          </cctvVideoService>
        </cctvCameraMetadataRecord>
      </cctvCameraList>
    </cctvSiteTablePublication>
  </genericPublicationExtension>
</payloadPublication>
```

```
</d2LogicalModel>
```

Listing 2: Situation extracted item

```
<d2LogicalModel xmlns="http://datex2.eu/schema/2/2_0"
   modelBaseVersion="2">
  <exchange>
    <supplierIdentification>
      <country>no</country>
      <nationalIdentifier>Norwegian Public Roads Administration</
         nationalIdentifier>
    </supplierIdentification>
  </exchange>
  <payloadPublication xmlns:xsi="http://www.w3.org/2001/XMLSchema
     -instance" lang="nob" xsi:type="SituationPublication">
    <publicationTime>2017-10-12T14:05:31+02:00</publicationTime>
    <publicationCreator>
      <country>no</country>
      <nationalIdentifier>Norwegian Public Roads Administration</
         nationalIdentifier>
    </publicationCreator>
    <situation id="NPRA_VL_180624" version="3">
      <overallSeverity>high</overallSeverity>
      <headerInformation>
        <confidentiality>noRestriction</confidentiality>
        <informationStatus>real</informationStatus>
        <urgency>normalUrgency</urgency>
      </headerInformation>
      <situationRecord id="NPRA_VL_180624_1" version="3" xsi:type
         ="RoadOrCarriagewayOrLaneManagement">
        <situationRecordCreationTime>2017-09-21T15:14:39+02:00</
           situationRecordCreationTime>
        <situationRecordVersionTime>2017-09-28T19:55:21+02:00</
           situationRecordVersionTime>
        <probabilityOfOccurrence>certain</probabilityOfOccurrence
```

```
          >
        <validity>
          <validityStatus>definedByValidityTimeSpec</
             validityStatus>
          <validityTimeSpecification>
            <overallStartTime>2017-10-08T20:00:00+02:00</
               overallStartTime>
            <overallEndTime>2018-02-10T06:00:00+01:00</
               overallEndTime>
          </validityTimeSpecification>
        </validity>
        <generalPublicComment>
          <comment>
            <values>
              <value>Toveis trafikk i ett l p p  grunn av
                 vegarbeid. Fare for k  .</value>
            </values>
          </comment>
          <commentType>description</commentType>
        </generalPublicComment>
        <generalPublicComment>
          <comment>
            <values>
              <value>Rv 162 Frederiks gate mellom Kristian IV s
                 gate og St. Olavs gate, i Oslo kommune</value>
            </values>
          </comment>
          <commentType>locationDescriptor</commentType>
        </generalPublicComment>
        <groupOfLocations xsi:type="Linear">
          <locationForDisplay>
            <latitude>59.917202</latitude>
            <longitude>10.735222</longitude>
          </locationForDisplay>
          <locationExtension>
            <locationExtension>
              <roadNumber>R162</roadNumber>
              <countyNumber>3</countyNumber>
              <municipalityNumber>300</municipalityNumber>
            </locationExtension>
          </locationExtension>
          <supplementaryPositionalDescription>
            <affectedCarriagewayAndLanes>
              <carriageway>mainCarriageway</carriageway>
              <lane>allLanesCompleteCarriageway</lane>
            </affectedCarriagewayAndLanes>
          </supplementaryPositionalDescription>
          <alertCLinear xsi:type="AlertCMethod4Linear">
            <alertCLocationCountryCode>F</
               alertCLocationCountryCode>
            <alertCLocationTableNumber>49</
               alertCLocationTableNumber>
            <alertCLocationTableVersion>5.1</
               alertCLocationTableVersion>
            <alertCDirection>
              <alertCDirectionCoded>both</alertCDirectionCoded>
            </alertCDirection>
            <alertCMethod4PrimaryPointLocation>
              <alertCLocation>
                <alertCLocationName>
                  <values>
                    <value lang="nob">St. Olavs gate</value>
                  </values>
                </alertCLocationName>
                <specificLocation>1115</specificLocation>
              </alertCLocation>
              <offsetDistance>
                <offsetDistance>0</offsetDistance>
              </offsetDistance>
            </alertCMethod4PrimaryPointLocation>
            <alertCMethod4SecondaryPointLocation>
              <alertCLocation>
                <alertCLocationName>
                  <values>
                    <value lang="nob">Nationaltheateret</value>
                  </values>
                </alertCLocationName>
                <specificLocation>1099</specificLocation>
              </alertCLocation>
              <offsetDistance>
                <offsetDistance>251</offsetDistance>
              </offsetDistance>
            </alertCMethod4SecondaryPointLocation>
          </alertCLinear>
          <linearExtension>
            <linearLineStringExtension>
              <gmlLineString srsName="EPSG:4326">
                <coordinates>10.734476395396888
                   59.91654782559496, 10.734851886829352
                   59.916864335670255, 10.735197440368598
                   59.91717993336038, 10.735660433176317
                   59.91756269724122, 10.735987491442804
                   59.91784664225683</coordinates>
              </gmlLineString>
            </linearLineStringExtension>
          </linearExtension>
        </groupOfLocations>
        <complianceOption>mandatory</complianceOption>
        <roadOrCarriagewayOrLaneManagementType>contraflow</
           roadOrCarriagewayOrLaneManagementType>
      </situationRecord>
      <situationRecord id="NPRA_VL_180624_2" version="3" xsi:type
```

```
 ="ConstructionWorks">
<situationRecordCreationTime>2017-09-21T15:14:39+02:00</
   situationRecordCreationTime>
<situationRecordVersionTime>2017-09-28T19:55:21+02:00</
   situationRecordVersionTime>
<probabilityOfOccurrence>certain</probabilityOfOccurrence
   >
<validity>
  <validityStatus>definedByValidityTimeSpec</
     validityStatus>
  <validityTimeSpecification>
    <overallStartTime>2017-10-08T20:00:00+02:00</
       overallStartTime>
    <overallEndTime>2018-02-10T06:00:00+01:00</
       overallEndTime>
  </validityTimeSpecification>
</validity>
<generalPublicComment>
  <comment>
    <values>
      <value>Toveis trafikk i ett l p p  grunn av
         vegarbeid. Fare for k  .</value>
    </values>
  </comment>
  <commentType>description</commentType>
</generalPublicComment>
<generalPublicComment>
  <comment>
    <values>
      <value>Rv 162 Frederiks gate mellom Kristian IV&
         apos; s gate og St. Olavs gate, i Oslo kommune</
         value>
    </values>
  </comment>
```

```
        <commentType>locationDescriptor</commentType>
      </generalPublicComment>
      <groupOfLocations xsi:type="Linear">
        <locationForDisplay>
          <latitude>59.917202</latitude>
          <longitude>10.735222</longitude>
        </locationForDisplay>
        <locationExtension>
          <locationExtension>
            <roadNumber>R162</roadNumber>
            <countyNumber>3</countyNumber>
            <municipalityNumber>300</municipalityNumber>
          </locationExtension>
        </locationExtension>
        <supplementaryPositionalDescription>
          <affectedCarriagewayAndLanes>
            <carriageway>mainCarriageway</carriageway>
            <lane>allLanesCompleteCarriageway</lane>
          </affectedCarriagewayAndLanes>
        </supplementaryPositionalDescription>
        <alertCLinear xsi:type="AlertCMethod4Linear">
          <alertCLocationCountryCode>F</
             alertCLocationCountryCode>
          <alertCLocationTableNumber>49</
             alertCLocationTableNumber>
          <alertCLocationTableVersion>5.1</
             alertCLocationTableVersion>
          <alertCDirection>
            <alertCDirectionCoded>both</alertCDirectionCoded>
          </alertCDirection>
          <alertCMethod4PrimaryPointLocation>
            <alertCLocation>
              <alertCLocationName>
                <values>
                  <value lang="nob">St. Olavs gate</value>
                </values>
              </alertCLocationName>
              <specificLocation>1115</specificLocation>
            </alertCLocation>
            <offsetDistance>
              <offsetDistance>0</offsetDistance>
            </offsetDistance>
          </alertCMethod4PrimaryPointLocation>
          <alertCMethod4SecondaryPointLocation>
            <alertCLocation>
              <alertCLocationName>
                <values>
                  <value lang="nob">Nationaltheateret</value>
                </values>
              </alertCLocationName>
              <specificLocation>1099</specificLocation>
            </alertCLocation>
            <offsetDistance>
              <offsetDistance>251</offsetDistance>
            </offsetDistance>
          </alertCMethod4SecondaryPointLocation>
        </alertCLinear>
        <linearExtension>
          <linearLineStringExtension>
            <gmlLineString srsName="EPSG:4326">
              <coordinates>10.734476395396888
                 59.91654782559496, 10.734851886829352
                 59.916864335670255, 10.735197440368598
                 59.91717993336038, 10.735660433176317
                 59.91756269724122, 10.735987491442804
                 59.91784664225683</coordinates>
            </gmlLineString>
          </linearLineStringExtension>
```

```
          </linearExtension>
        </groupOfLocations>
        <constructionWorkType>roadImprovementOrUpgrading</
           constructionWorkType>
      </situationRecord>
    </situation>
  </payloadPublication>
</d2LogicalModel>
```

Listing 3: Weather Measured Data extracted item

```
<d2LogicalModel xmlns="http://datex2.eu/schema/2/2_0"
   modelBaseVersion="2">
  <exchange>
    <supplierIdentification>
      <country>no</country>
      <nationalIdentifier>Norwegian Public Roads Administration</
         nationalIdentifier>
    </supplierIdentification>
  </exchange>
  <payloadPublication xmlns:xsi="http://www.w3.org/2001/XMLSchema
     -instance" lang="nob" xsi:type="MeasuredDataPublication">
    <publicationTime>2017-10-12T14:08:07.559+02:00</
       publicationTime>
    <publicationCreator>
      <country>no</country>
      <nationalIdentifier>Norwegian Public Roads Administration</
         nationalIdentifier>
    </publicationCreator>
    <measurementSiteTableReference id="WOST" targetClass="
       MeasurementSiteTable" version="20171012133922000"/>
    <headerInformation>
      <confidentiality>noRestriction</confidentiality>
      <informationStatus>real</informationStatus>
    </headerInformation>
    <siteMeasurements>
      <measurementSiteReference id="242" targetClass="
         MeasurementSiteRecord" version="10"/>
      <measurementTimeDefault>2017-10-12T14:00:00.000+02:00</
         measurementTimeDefault>
      <measuredValue index="201">
        <measuredValue>
          <basicData xsi:type="HumidityInformation">
            <humidity>
              <relativeHumidity>
                <percentage>69.0</percentage>
              </relativeHumidity>
            </humidity>
          </basicData>
        </measuredValue>
      </measuredValue>
      <measuredValue index="2501">
        <measuredValue>
          <basicData xsi:type="PrecipitationInformation">
            <precipitationDetail>
              <precipitationIntensity>
                <millimetresPerHourIntensity>0.0</
                   millimetresPerHourIntensity>
              </precipitationIntensity>
            </precipitationDetail>
          </basicData>
        </measuredValue>
      </measuredValue>
      <measuredValue index="801">
        <measuredValue>
          <basicData xsi:type="RoadSurfaceConditionInformation">
            <roadSurfaceConditionMeasurements>
              <roadSurfaceTemperature>
                <temperature>13.5</temperature>
              </roadSurfaceTemperature>
            </roadSurfaceConditionMeasurements>
          </basicData>
        </measuredValue>
      </measuredValue>
      <measuredValue index="101">
        <measuredValue>
          <basicData xsi:type="TemperatureInformation">
            <temperature>
              <airTemperature>
                <temperature>10.5</temperature>
              </airTemperature>
            </temperature>
          </basicData>
        </measuredValue>
      </measuredValue>
      <measuredValue index="301">
        <measuredValue>
          <basicData xsi:type="TemperatureInformation">
            <temperature>
              <dewPointTemperature>
                <temperature>5.1</temperature>
              </dewPointTemperature>
            </temperature>
          </basicData>
        </measuredValue>
      </measuredValue>
      <measuredValue index="1401">
        <measuredValue>
          <basicData xsi:type="VisibilityInformation">
            <visibility>
              <minimumVisibilityDistance>
                <integerMetreDistance>9999</integerMetreDistance>
              </minimumVisibilityDistance>
```

```
            </visibility>
          </basicData>
        </measuredValue>
      </measuredValue>
    </siteMeasurements>
  </payloadPublication>
</d2LogicalModel>
```

3.2 JSON messages format

Following the general structure already explained in *Deliverable D3.2 - Publish/Subscribe and security functionality*, in this section are reported example messages that end users (i.e., Data Consumer or Data Producer) exchange in order to publish or subscribe for data, and which kind of messages they expect to receive after performing these operations. To provide an example, it is assumed that a Data Producer generates content under the topic */bv/nametest*, which contains a random number and a timestamp.

In order to publish the initial content and then announce the related punctual update, the Data Producer need to format the messages as follow:

Listing 4: INIT content publication

```
{"header":"bv/nametest/init",
"type":"publish",
"content":{}
}
```

Listing 5: UPDATE content publication

```
{"header":"bv/nametest/update",
"type":"publish",
"content":{}
}
```

In this case, the *content* field is left empty because no data is sent to the Producer Proxy. As soon as these messages are received, the Producer Proxy proceeds to announce the new/updated data to IRN and, if any, to subscribed users.

The messages sent and received in the case of subscription for the given name are reported below:

Listing 6: Subscription for a content

```
{"header":"subscribe",
"type":"data",
"content":{"namelist":"bv/nametest"}
}
```

As the user wants to retrieve a specific name, it needs to include it in additional field *namelist* under the *content* field.

Listing 7: Data message received

```
{"header":"bv/nametest",
"type":"data",
"content":{
"data":{"timestamp":1491825767482,"number":56"}
}
}
```

More details about these messages can be found in *Deliverable D6.1 - Technology dependent interfaces.*

3.3 Processing of DATEX II data and pub-sub process

Original DATEX II files are processed by the Data Producer as described in Figure 19. In details, the operations are explained in the following:

1. First of all, data are fetched from the NPRA server via https, every 3 minutes.
2. Downloaded XML files are parsed in order to identify geographical information associated to each record.
3. Records belonging to the same tile are grouped within a single travel-centric content.
4. Afterwords, a new XML file, formatted according to the DATEX II standard, is created for each tile.
5. The outcomes represents the initial data, announced with the *front-end API* namely *Producer_Publish_Init* to the platform.
6. Then contents are compared with respect to those generated at the previous cycle. If an update is found, the Data Producer announces such a content to the Producer Proxy

using the *front-end API* namely *Producer_Publish_Update* Application Programming Interface (API).

7. Finally, the Data Producer waits for future content requests coming from subscribers.

8. The new download is scheduled after 3 minutes.

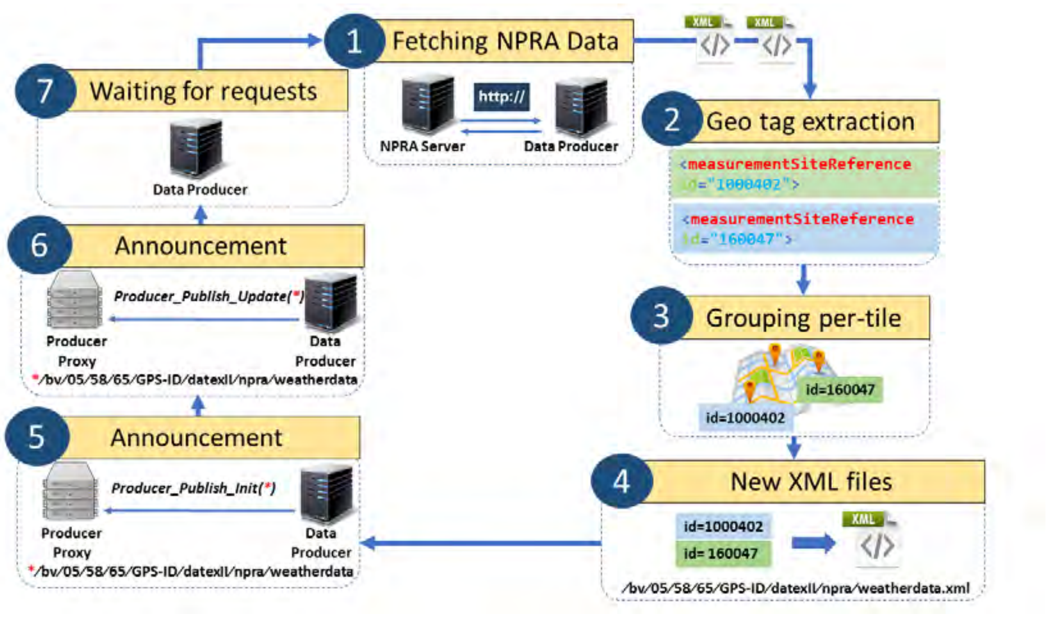


Figure 19: Processing of DATEX II files

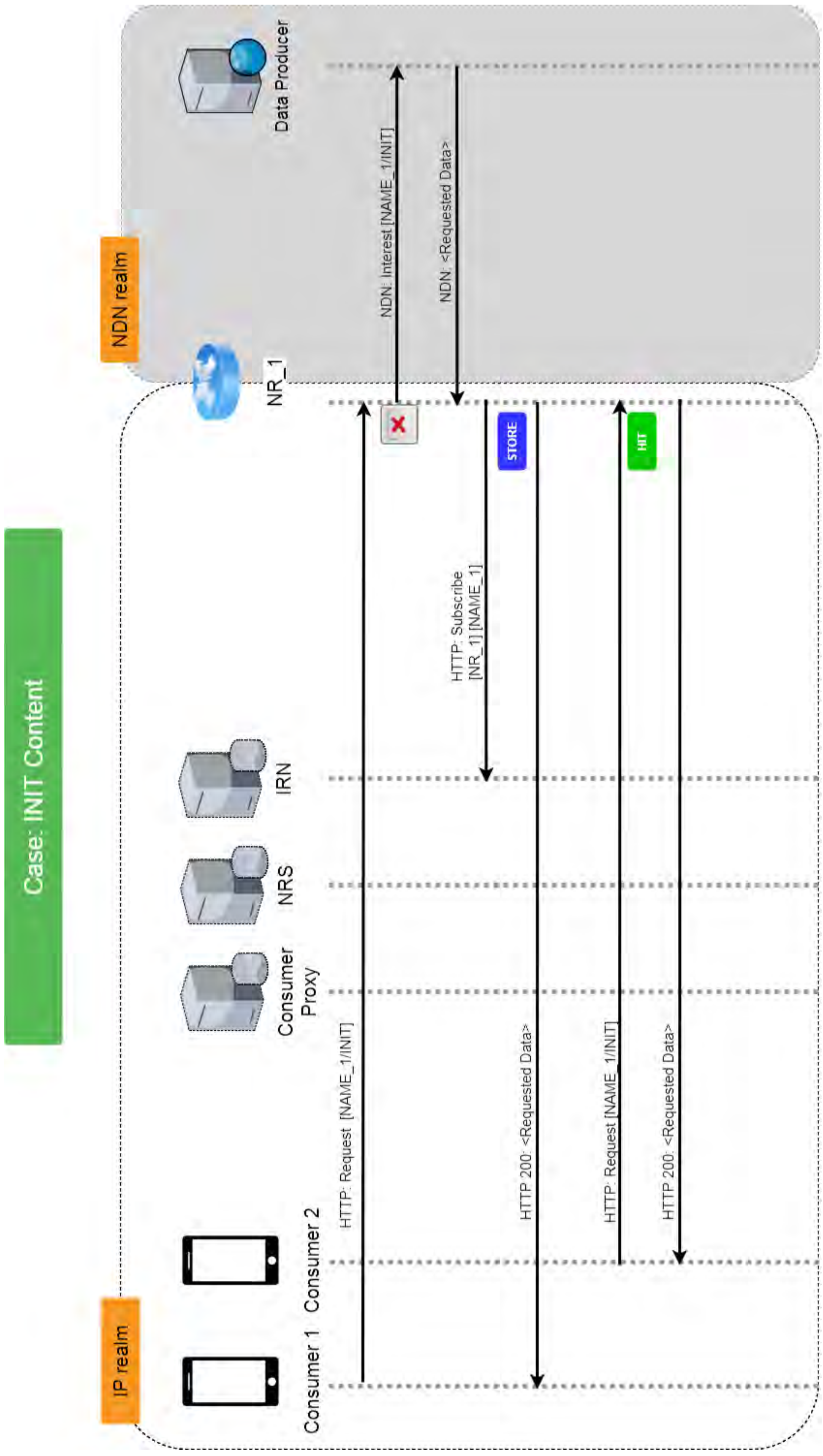
The script parsing NPRA's files is reported in Section B. Specific comments are dedicated to the executed operations.

3.4 Customized caching strategies

When the content is delivered to the Data Consumer, it may be temporarily cached in intermediate Named Routers, allowing them to satisfy future requests for the same content. To this end, the BONVOYAGE Communication System integrates a customized caching strategy for DATEX II data. The leading concept is to follow a different management of cached resources, based on the type of contents they refer to.

The developed algorithm is summarized in the sequence diagrams reported in Figure 20 and Figure 21. It mainly embraces three possible situations.

The first case is verified when an INIT content is requested from the Data Consumer, but it is not available in intermediate caches. The request is forwarded by Internames to the Data Producer, that will answers with the corresponding content. Then, that content is cached in intermediated Named Routers. In order to learn about future updated for the same information,



Case: INIT Content
IP realm
NDN realm
Consumer 1
Consumer 2
Consumer Proxy
NRS
IRN
NR_1
Data Producer
HTTP: Request [NAME_1/INIT]
NDN: Interest [NAME_1/INIT]
NDN: <Requested Data>
HTTP: Subscribe [NR_1] [NAME_1]
STORE
HTTP 200: <Requested Data>
HTTP: Request [NAME_1/INIT]
HIT
HTTP 200: <Requested Data>

Figure 20: Caching strategies for DATEX II contents

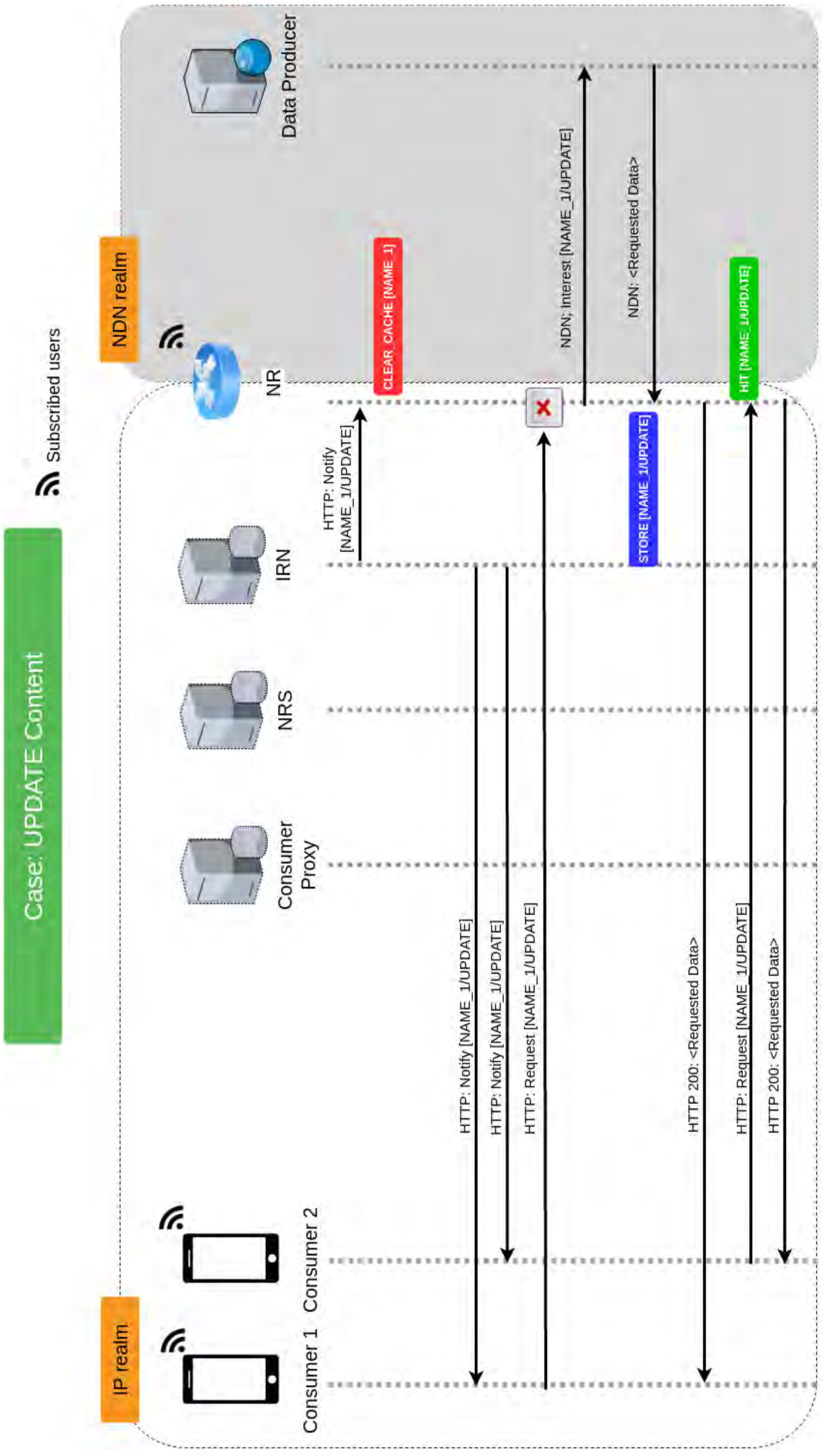

Case: UPDATE Content
Subscribed users
IP realm
NDN realm
Consumer 1
Consumer 2
Consumer Proxy
NRS
IRN
NR
Data Producer
HTTP: Notify [NAME_1/UPDATE]
CLEAR_CACHE [NAME_1]
HTTP: Notify [NAME_1/UPDATE]
HTTP: Notify [NAME_1/UPDATE]
HTTP: Request [NAME_1/UPDATE]
NDN: Interest [NAME_1/UPDATE]
NDN: <Requested Data>
STORE [NAME_1/UPDATE]
HTTP 200: <Requested Data>
HTTP: Request [NAME_1/UPDATE]
HIT [NAME_1/UPDATE]
HTTP 200: <Requested Data>

Figure 21: Caching strategies for DATEX II contents

also intermediate Named Routers (like the Producer Proxy) issue a subscription request to Internames Rendezvous Node (IRN). The second case is verified when INIT or UPDATE contents are requested by the Consumer Proxy and a match is found in an intermediated cache. Now, the router simply reads the content from the cache and sends it back to the Consumer Proxy. The third case is verified when the Data Producer announces the availability of an UPDATE content. Here, the router is notified (as other subscribers) and INIT and UPDATE contents stored in the cache are deleted because they became obsolete. Thus, when a new request arrives, it is forwarded to the remote Data Producer. Once receive a new data, the router caches again the content and sends it backwards to the subscriber.

3.5 Developed testbed

In general, the BONVOYAGE Communication System can be used to disseminate any kind of contents. Data Producers, however, must properly arrange their data, thus making their representation in line with the design principles presented in the previous Sections. On the other hand, Data Consumers interested in custom-formatted data, must have knowledge about their representation, to properly parse and process the information in high-level applications.

This section introduces an experimental testbed that focuses on implementing and testing the Publish-Subscribe functionalities, in order to disseminate a customized data by jointly exploiting the Proxies, the APIs and the middleware features exposed so far.

According to the Big Picture of the BONVOYAGE Communication System, a heterogeneous network scenario has been set up, as shown in Figure 22. The implemented entities are:

- Named Routers that interconnect an Named-Data Networking (NDN) realm, namely **Net_3** (that acts as Core Network), with two IP realms, namely **Net_1** and **Net_2**, respectively;
- Name Resolution Services for path discovery and routing operations across the aforementioned realms;
- Consumer and Producer proxies connected to the IP realms, that expose WebSocket endpoints to Data Consumers/Producers;
- An instance of Data Producer and Data Consumer that generates formatted data, publishes them in the BONVOYAGE platform, subscribes to the related topic and retrieves the init and updates contents, by using the front-end APIs.

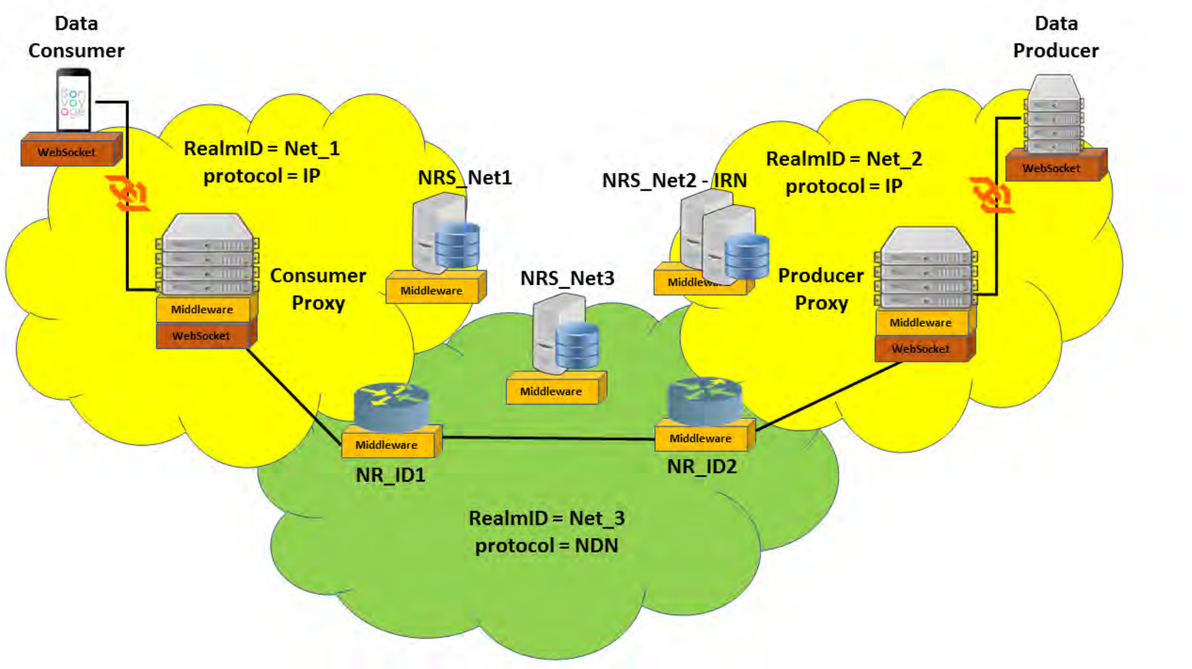


Figure 22: Reference scenario implemented

An example XML file generated from the processing routine and delivered to Data Consumer is shown below.

3.6 Performance evaluation

A preliminary performance assessment of the BONVOYAGE Communication System is discussed herein. Specifically, tests are conducted by using the experimental testbed developed in the context of the BONVOYAGE project.

At the beginning, the performance gain offered by the BONVOYAGE Communication System with respect to the baseline usage of data exposed by the NPRA server is evaluated. To this end, it is assumed that a Data Consumer is interested to retrieve contents available in a geographical area that includes Oslo city. The GPS coordinates for the low-left and high-right points of that area are: (latitude = 6, longitude = 58) and (latitude = 12, longitude = 63), respectively. Figure 23 shows the aggregate number of contents (e.g., portion of the XML file associated to a given tile) downloaded by the Data Consumer during the time. Results clearly demonstrate that the BONVOYAGE Communication System is able to significantly reduce the amount of data processed by the Data Consumer. It, in fact, delivers only the updated portions of the original XML files that are of interest for the Data Consumer. This important behavior also leads to a reduction of the average bandwidth consumption at the Data Consumer side. The experimental test demonstrates that a Data Consumer directly connected to the NPRA server consumes an average bandwidth equal to 424.93 kbps. On the contrary, when the Data Consumer is attached to the BONVOYAGE Communication System, an average bandwidth consumption of 19.96 kbps is just required. Therefore, a performance gain of an order of magnitude is reached.

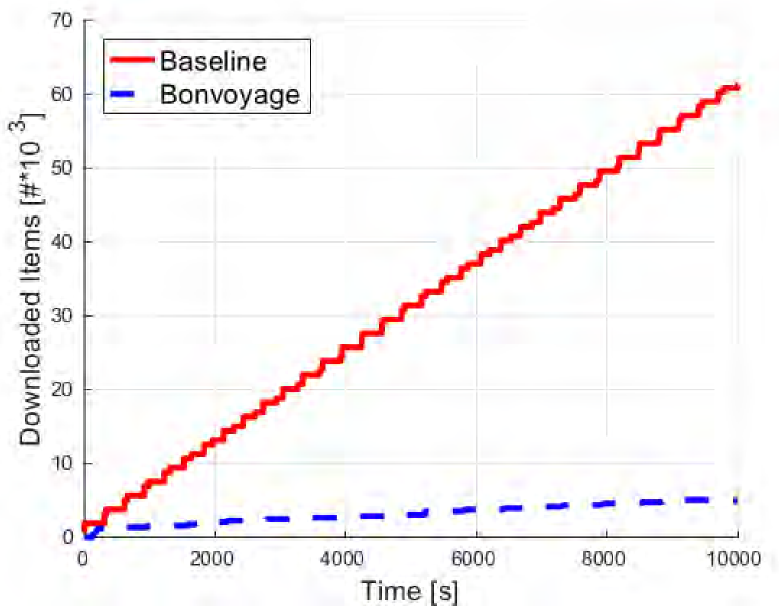
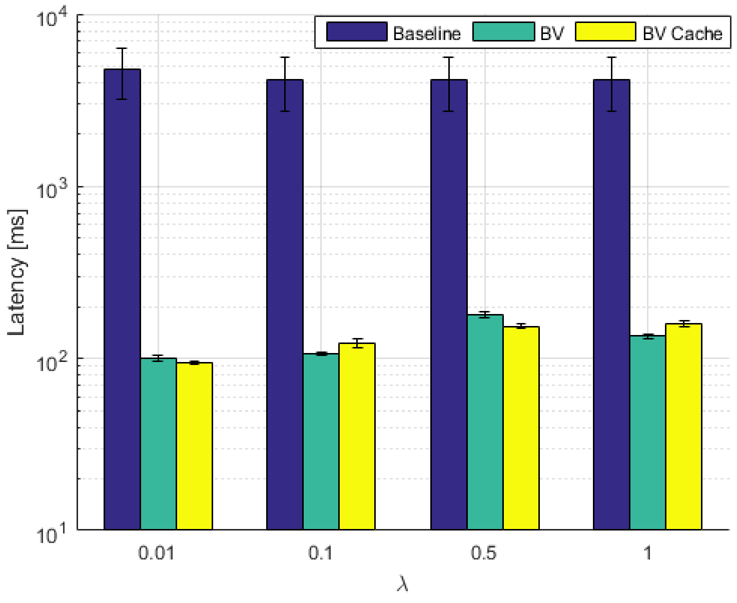


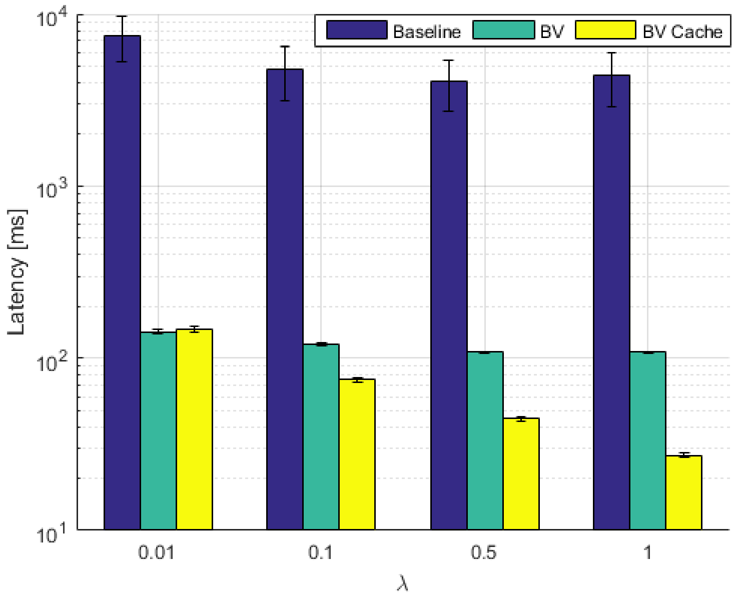
Figure 23: Number of downloaded DATEX II records during the time

The impact that number of Data Consumers and caching strategy have on the performance of the BONVOYAGE Communication System is evaluated too. In this case, the conducted test considers a number of Data Consumers that join the BONVOYAGE platform according to a Poisson Distribution with parameter $\lambda \in [0.01, 1]$ and ask for the same set of contents. The whole system is observed for 300 s and reported results are averaged among 15 different seeds. In this case, the selected area can range from the 10% and the 100% of the whole Norway. These points have been chosen after an empirical analysis of the useful items on the NPRA server. For the sake of simplicity, the spatial distribution of items in this area is assumed to be uniform.

Figure 24 shows the measured communication latencies. The amount of network traffic in the baseline case results in higher values, as the users are required to directly interact with the server to ask for every file everytime data are needed. As a consequence, the traffic throughout the whole network increases. The BONVOYAGE Communication System, instead, is able to manage the aforementioned files in order to make INIT and Update files available for subscribed users as long as they are the most updated ones. Moreover, one of the most interesting results comes out from the introduction of caching strategy, which empowers the noticeable results registered when using the BONVOYAGE Communication System. In fact, it is registered a significant reduction of latencies. As a consequence, it is possible to ensure the best Quality of Service (QoS) level offered to Data Consumers.



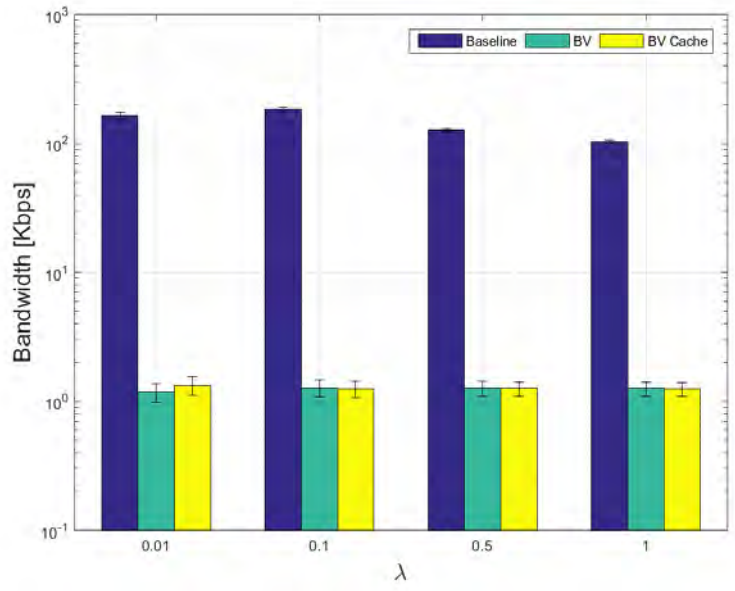
(a)



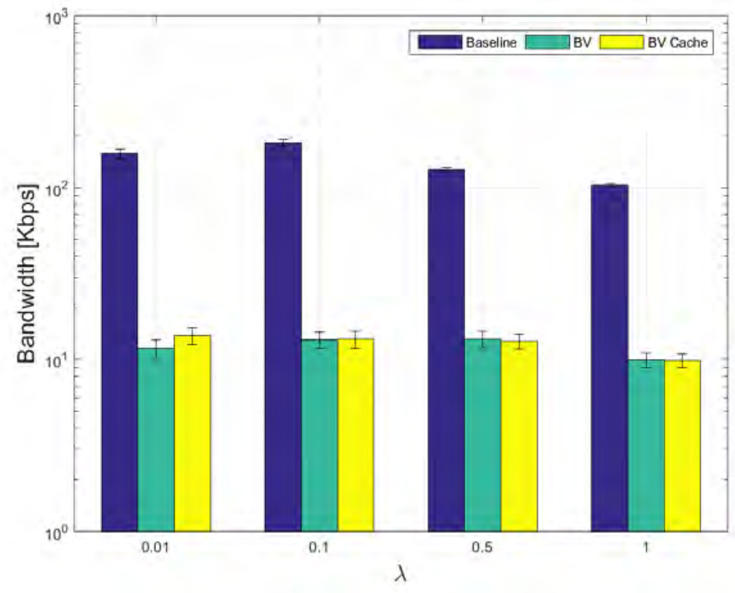
(b)

Figure 24: Communication latencies in case (a) 10% and (b) 100%

In Figure 25 is reported the comparison for the single client bandwidth consumption. Thanks to the processing and the publish subscribe mechanism over WebSocket, the BONVOYAGE Communication System can significantly reduce the amount of data delivered, by retrieving the same number of contents.



(a)



(b)

Figure 25: Single client bandwidth consumption in case (a) 10% and (b) 100%

The cache size is shown in Figure 26. Results demonstrate that the customized caching strategy is able to achieve a good performance when the system scales up, in terms of number of contents requested. Therefore, values reported never exceed the theorical estimed size for cache the whole set of contents, and the highest value is 85% lower of the upper bound. The latter consideration results from the worst case estimation. In fact, considering all the available names published in files of the same size, equal to the higher measured value (30 kB), the overall resulting size would be close to 400 MB.

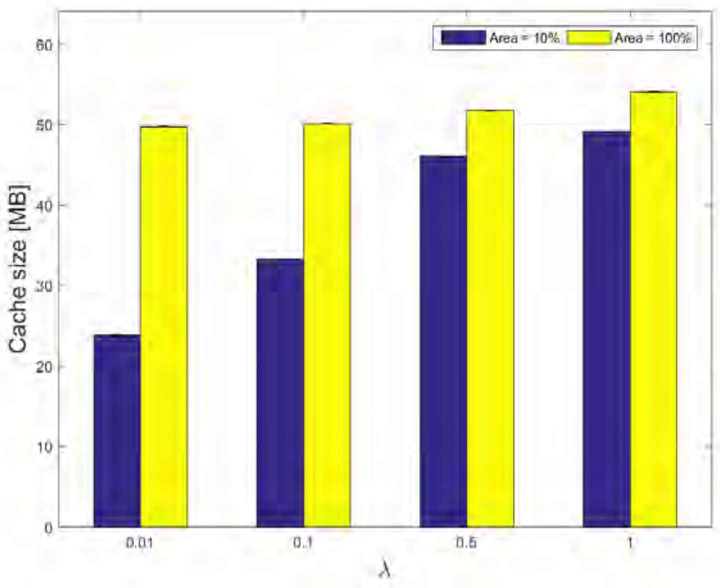


Figure 26: Cache size

As a preliminary consideration, it is possible to say that the higher the number of contents made available for subscribers, the higher the hit rate would be. In a similar fashion, it is reasonable that as the number of requests performed by subscribed users increases, so will do the cache hit rate. To prove these assumption, a cache hit rate measurement has been performed. In Figure 27, the cache hit rate is, then, shown. Given the performed aggregation activity, as expected, the hit rate increases according to the number of requested contents. Measured values can reach up to the 80% of contents served by the cache.

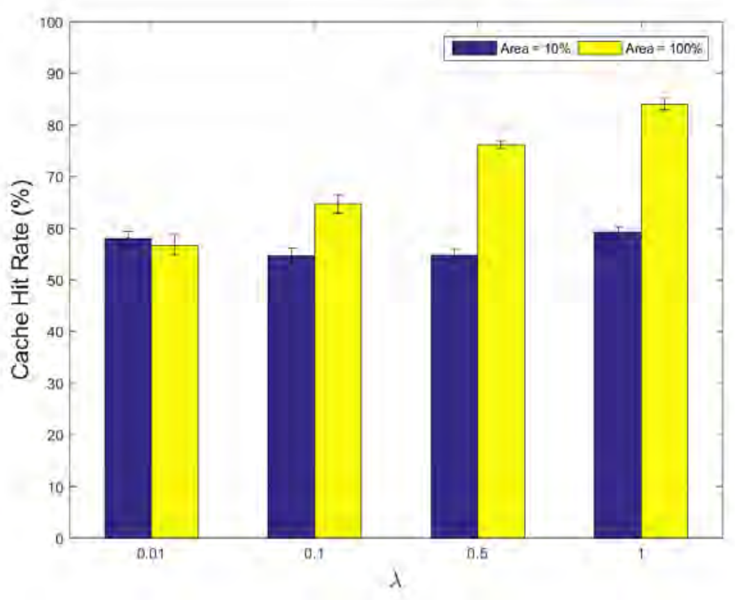


Figure 27: Cache hit rate

3.7 Networking Security Functionalities

To guarantee that communication between entities are secure and certified, security functionalities are implemented to assure restricted messages exchange. Regarding the Core Network, it is released as an overlay, private and closed system. Thus, in the envisoned architeture it has a built-in security, and nodes communicates each other and are connected to the Core Network within a Virtual Private Network (VPN) technology.

The secure connection between end users (Data Consumer and Data Producers) and the Proxy endpoints is performed thanks to WebSocket Secure (WSS) technology over the Transport Layer Security (TLS) version 1.2. The Subscription requests performed by the Data Consumers have to be authenticated. In order to guarantee both authenticity and integrity of the communicated data, the envisioned solution is the usage of the HASH of the data signed with a private key. All the entities involved in the communication processes have a certificate, which is released by the BONVOYAGE Certification Authority.

References

[1] A. Detti, N. B. Melazzi, M. Orru, R. Paolillo, and G. Rossi, "Opengeobase: Information centric networking meets spatial database applications," in *2016 IEEE Globecom Workshops (GC Wkshps)*, Dec 2016, pp. 1–7.

[2] A. Detti, M. Orru, R. Paolillo, G. Rossi, P. Loreti, L. Bracciale, and N. B. Melazzi, "Application of information centric networking to nosql databases: The spatio-temporal use case," in *2017 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, June 2017, pp. 1–6.

[3] R. H. Güting, "An introduction to spatial database systems," *The VLDB JournalThe International Journal on Very Large Data Bases*, vol. 3, no. 4, pp. 357–399, 1994.

[4] "Mongodb," http://www.mongodb.com.

[5] G. Grassi, D. Pesavento, G. Pau, L. Zhang, and S. Fdida, "Navigo: Interest forwarding by geolocations in vehicular named data networking," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2015 IEEE 16th International Symposium on a*. IEEE, 2015, pp. 1–10.

[6] W. Drira and F. Filali, "Ndn-q: An ndn query mechanism for efficient v2x data collection," in *Sensing, Communication, and Networking Workshops (SECON Workshops), 2014 Eleventh Annual IEEE International Conference on*. IEEE, 2014, pp. 13–18.

[7] "Geojson," http://geojson.org.

[8] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*. ACM, 2009.

[9] A. Hoque, S. O. Amin, A. Alyyan, B. Zhang, L. Zhang, and L. Wang, "Nlsr: named-data link state routing protocol," in *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking*. ACM, 2013.

[10] A. Melo-Castillo, P. Bure, L. F. Herrera-Quintero, and K. Banse, "Design and implementation of datex ii profiles for truck parking systems," in *2017 15th International Conference on ITS Telecommunications (ITST)*, May 2017, pp. 1–7.

[11] N. B. Melazzi, A. Detti, M. Arumaithurai, and K. Ramakrishnan, "Internames: A name-to-name principle for the future internet," *10th International Conference on Heterogeneous*

Networking for Quiteuality, Reliability, Security and Robustness (QShine), pp. 146–151, 2014.

[12] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, k. claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named data networking," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 66–73, Jul. 2014.

[13] "The Named Data Networking project," 2016. [Online]. Available: http://www.named-data.net/

[14] T. Hardie, "Clarifying registry procedures for the websocket subprotocol name registry," Tech. Rep., 2016.

Appendix

A Internames background

The BONVOYAGE Communication System, targets the efficient and scalable dissemination of travel-centric (or ITS) contents across an heterogeneous network infrastructure. Specifically, it integrates *Internames*, e.g., a novel network architecture evolving from Information-Centric Networking (ICN) and designed to allow name-to-name communications in heterogeneous systems made up by different realms working with various networking protocols [11]. In the proposed solution, the core network of Internames is based on a concrete ICN implementation, namely NDN [12][13]. Deployed on large scale as an overlay network, some of NDN nodes act as border routers and offer a point of access to a specific portion of the current Internet, based on the IP protocol (see Figure 28). Thus, Internames is in charge of offering low-level networking functionalities for enabling the interaction among IP and NDN network realms (as discussed in [11]).

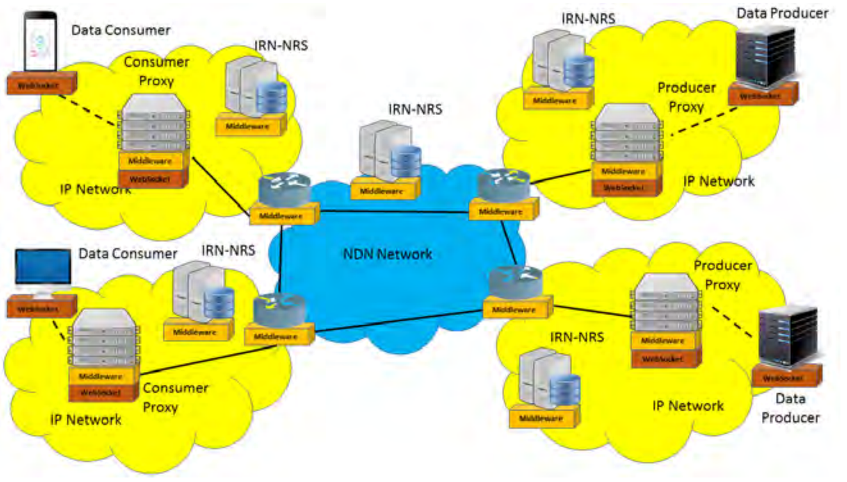


Figure 28: Internames architecture

The whole architecture depicted in Figure 28 integrates the following entities:

- **Data Consumer**: end user willing to retrieve travel-centric information for planning and optimizing transportation services;
- **Data Producer**: entity that generates travel-centric contents and expose them within the platform;
- **Consumer Proxy**: logical node offering a standardized interface between Data Consumers and the rest of the BONVOYAGE Communication System;

- **Producer Proxy**: logical node offering a standardized interface between Data Producer and the rest of the BONVOYAGE Communication System;
- **Named Router**: border router connecting two or more network realms;
- **Name Resolution Service (NRS)**: main entity of the Internames architecture, handling inter-realm routing operations (as discussed in [11]);
- **Internames Rendezvous Node (IRN)**: new entity added to the Internames for handling publish-subscribe functionalities.

The dissemination of travel-centric contents across the heterogeneous network architecture is performed according to both *request-response* and *publish-subscribe* communication schema. The interaction between Data Consumer and Consumer Proxy, as well as between Data Producer and Producer Proxy, is allowed through a specific set of *front-end APIs.* In Internames, instead, logical nodes interact with each other by means of *networking APIs.*

Logical nodes involved in Internames, that are Consumer Proxy, Producer Proxy, Named Router, NRS and IRN, are deployed within an heterogeneous network architecture. Therefore, the BONVOYAGE Communication System aims at easing their interaction during the provisioning of both request-response and publish-subscribe communication schema, as well as making the implementation of high-level functionalities independent from the underlying communication technology. To this end, *networking APIs* are exposed by a distributed middleware layer, namely Internames Service Layer. These APIs are: $Netw\_Request$, $Netw\_Subscribe$, $Netw\_Del\_Subscribe$, $Netw\_Announce$, $Netw\_Del\_Announce$, $Netw\_Notify$. Specifically, each *networking API* triggers the execution of one or more atomic networking operations, whose implementation depends on the underlying communication technology.

The $Netw\_Request$ API is used to retrieve a travel-centric content. In the case the requesting node (i.e., the Data Consumer or any Named Router entity in Internames) is directly connected to the realm specified in the name of the content, the considered data is immediately fetched from the network. Otherwise, the NRS node is contacted for driving the inter-realm routing of the request. With the $Netw\_Subscribe$ API, the Data Consumer sends subscription requests to the reference IRN entity of Internames. Such a request can be canceled by using the $Netw\_Del\_Subscribe$ API. The Producer Proxy announces the availability of new content to the IRN entity of Internames by using the $Netw\_Announce$ API. The $Netw\_Del\_Announce$ API, instead, is used to declare that the content is no longer available in the system. Finally, when IRN realizes the availability of a new version of a travel-centric content, it sends a notification to all the subscribers by using the $Netw\_Notify$ API. Then, subscribers (i.e., Data Consumers) may retrieve the content by means of the *Netw_Request* API, as it has been previously described.

More details about these APIs can be found in *Deliverable D3.2 - Publish/Subscribe and security functionality.*

A.1 Front-end APIs for consumers and producers

Data Producer and Data Consumer are not directly involved in Internames. They, instead, interact with Producer Proxy and Consumer Proxy, respectively, through *front-end APIs*. To support publish-subscribe functionalities, these APIs leverage the WebSocket [14] technology, useful to establish an asynchronous and bi-directional connection between the communicating entities. In summary, *front-end APIs* include: *Producer_Publish_Init*, *Producer_Publish_Update*, *Consumer_Request*, and *Consumer_Subscribe*.

For each travel-centric content, the Data Producer generates initial data (namely *INIT*) and punctual updates (namely *UPDATE*). Initial data are delivered to the Data Consumer as soon as it joins the platform and makes a request or a subscription. Punctual updates, instead, refer to any single modification of the data made available by the Data Producer, and are delivered to subscribers when they are generated. Initial data and punctual updates are announced by the Data Producer to the reference Producer Proxy by using *Producer_Publish_Init* and *Producer_Publish_Update* APIs, respectively. Then, the Producer Proxy announces to the BONVOYAGE Communication System the availability of a new version of the data. To this end, it uses a *networking API*, namely *Netw_Announce*, exposed by Internames.

The Data Consumer can request a given travel-centric content or make a subscription request by using *Consumer_Request* or *Consumer_Subscribe* APIs, respectively. Both of these APIs, receive in input the list of names associated to the contents of interest for the Data Consumer. The list of names can be retrieved by contacting serch engines, like OpenGeoBase [1]. Once the Data Consumer calls the *Consumer_Request* API, the Consumer Proxy immediately retrieves requested contents by using a *networking API*, namely *Netw_Request*, exposed by Internames. When the Data Consumer calls the *Consumer_Subscribe* API, instead, the Consumer Proxy executes more complex tasks. First, as soon as the Consumer Proxy receives the list of names from the Data Consumer, it immediately retrieves initial data associated to the considered contents. The *networking API*, namely *Netw_Request*, is used for this purpose. Then, to reduce the traffic load in the core network of Internames, the Consumer Proxy manages an aggregation of subscription requests. Indeed, if any further subscription for the same contents was already processed in the past, the Consumer Proxy makes a subscription request to the BONVOYAGE

Communication System by using another *networking API*, namely *Netw_Subscribe*, still exposed by Internames. Otherwise, it just updates the list of Data Consumers interested to some contents. The subscription request is deleted when the Data Consumer closes the WebSocket connection with the Consumer Proxy.

To reach efficient, asynchronous, and decoupled communications in the front-end, messages exchanged between Data Consumer and Consumer Proxy, as well as between Data Producer and Producer Proxy, are encoded with JSON format. In particular, they include three fields: header, type, and content. The header contains general information, like the name of the published/requested content. This part is mandatory in order to disambiguate communications and properly address the message to the user/consumer/application who/which asked for it. The type field specifies the type of message (for instance, whether it contains logging information or data). Finally, the content filed stores the real data requested by the consumer.

More details about these APIs can be found in *Deliverable D3.2 - Publish/Subscribe and security functionality*.

B DATEX II Listing

```
package it.telematics.isl.DatexIIGateway;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashSet;
import java.util.concurrent.atomic.AtomicInteger;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.TransformerException;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;
```

```java
import it.telematics.isl.bonvoyage.NamesGenerator.NamesGenerator;
import it.telematics.isl.bonvoyage.XMLUtils.XMLUtils;
import it.telematics.isl.core.utils.FileUtils;

public class DatexIINameAnalyzer {
private static final float side = 0.1f;// 10Km^2
private static Pattern    geo_tag_rgx = Pattern.compile("([^\\p{
   Alpha} \\p{Punct}][0-9]+)");
private static Pattern    diff_rgx = Pattern.compile("([^,]+)
   ,([^,]+),([^,]+)\n");

public static HashSet<String> analyzeTable(String table, Document
    newDoc, Document oldDoc, String workingDir)
throws IOException, SAXException, ParserConfigurationException,
   TransformerException, InterruptedException {
HashSet<String> namesList = new HashSet<String>();

// create directory
new File(workingDir).mkdirs();

// create util files
File idsF = new File(workingDir + "/ids.txt");
File diffsF = new File(workingDir + "/diffs.txt");
StringBuilder ids = new StringBuilder();
StringBuilder diffs = new StringBuilder();

// retrieve common tag
String[] d2LogicalModel = XMLUtils.toString(XMLUtils
.extractElementsByTagName(newDoc.getDocumentElement(), "
   d2LogicalModel")
.get(0)
.cloneNode(false)).replace("><", ">\n<").split("\n");
FileUtils.printOnFile(d2LogicalModel[0],
new File(workingDir + "/d2LogicalModelInit"));
FileUtils.printOnFile(d2LogicalModel[1],
new File(workingDir + "/d2LogicalModelEnd"));

FileUtils.printOnFile(XMLUtils.toString(XMLUtils
.extractElementsByTagName(newDoc.getDocumentElement(), "exchange"
   ).get(0)),
new File(workingDir + "/exchange"));

String[] payloadPublication = XMLUtils.toString(XMLUtils
.extractElementsByTagName(newDoc.getDocumentElement(), "
   payloadPublication")
.get(0)
.cloneNode(false)).replace("><", ">\n<").split("\n");
FileUtils.printOnFile(payloadPublication[0],
new File(workingDir + "/payloadPublicationInit"));
FileUtils.printOnFile(payloadPublication[1],
new File(workingDir + "/payloadPublicationEnd"));

FileUtils.printOnFile(XMLUtils.toString(XMLUtils
.extractElementsByTagName(newDoc.getDocumentElement(), "
   publicationTime").get(0)),
new File(workingDir + "/publicationTime"));

FileUtils.printOnFile(XMLUtils.toString(XMLUtils
.extractElementsByTagName(newDoc.getDocumentElement(), "
   publicationCreator").get(0)),
new File(workingDir + "/publicationCreator"));


if (table.endsWith("GetCCTVSiteTable")) {

// extract custom tag
FileUtils.printOnFile(XMLUtils.toString(XMLUtils
.extractElementsByTagName(newDoc.getDocumentElement(), "
    genericPublicationName").get(0)),
new File(workingDir + "/genericPublicationName"));

String[] genericPublicationExtension = XMLUtils.toString(XMLUtils
.extractElementsByTagName(newDoc.getDocumentElement(), "
    genericPublicationExtension")
.get(0)
.cloneNode(false)).replace("><", ">\n<").split("\n");
FileUtils.printOnFile(genericPublicationExtension[0],
new File(workingDir + "/genericPublicationExtensionInit"));
FileUtils.printOnFile(genericPublicationExtension[1],
new File(workingDir + "/genericPublicationExtensionEnd"));

String[] cctvSiteTablePublication = XMLUtils.toString(XMLUtils
.extractElementsByTagName(newDoc.getDocumentElement(), "
    cctvSiteTablePublication")
.get(0)
.cloneNode(false)).replace("><", ">\n<").split("\n");
FileUtils.printOnFile(cctvSiteTablePublication[0],
new File(workingDir + "/cctvSiteTablePublicationInit"));
FileUtils.printOnFile(cctvSiteTablePublication[1],
new File(workingDir + "/cctvSiteTablePublicationEnd"));

FileUtils.printOnFile(XMLUtils.toString(XMLUtils
.extractElementsByTagName(newDoc.getDocumentElement(), "
    headerInformation").get(0)),
new File(workingDir + "/headerInformation"));

String[] cctvCameraList = XMLUtils.toString(XMLUtils.
    extractElementsByTagName(newDoc.getDocumentElement(), "
    cctvCameraList")
.get(0)
.cloneNode(false)).replace("><", ">\n<").split("\n");
```

```
FileUtils.printOnFile(cctvCameraList[0],
new File(workingDir + "/cctvCameraListInit"));
FileUtils.printOnFile(cctvCameraList[1],
new File(workingDir + "/cctvCameraListEnd"));

FileUtils.printOnFile(XMLUtils.toString(XMLUtils
.extractElementsByTagName(newDoc.getDocumentElement(), "
   cctvCameraListVersionTime").get(0)),
new File(workingDir + "/cctvCameraListVersionTime"));

// need to set id for every element
if (oldDoc != null) {
NodeList nodeList = oldDoc.getElementsByTagName("
   cctvCameraMetadataRecord");
for (int j = 0; j < nodeList.getLength(); j++) {
Element node = (Element) nodeList.item(j);
// need to set id as ID
node.setIdAttribute("id", true);
}
}

// process
ArrayList<Element> recordList = XMLUtils.extractElementsByTagName
   (newDoc.getDocumentElement(),
"cctvCameraMetadataRecord");

// stats

for (Element el1 : recordList) {

// extract geo-tagged info
String id  = el1.getAttribute("id");
float longitude = new Float(XMLUtils.extractElementsByTagName(el1
   , "longitude")
```

```
.get(0).getTextContent());
float latitude = new Float(XMLUtils.extractElementsByTagName(el1,
    "latitude")
.get(0).getTextContent());

// flush node to file
FileUtils.printOnFile(XMLUtils.toString(el1.cloneNode(true)),
new File(workingDir + "/" + id));

// update ids
ids.append(id);
ids.append(",");
ids.append(latitude);
ids.append(",");
ids.append(longitude);
ids.append("\n");

if (oldDoc != null) {

Element el2 = oldDoc.getElementById(id);

if (el2 == null ||
!XMLUtils.toString(el1.cloneNode(true))
.equals(XMLUtils.toString(el2.cloneNode(true)))) {
// update diffs
diffs.append(id);
diffs.append(",");
diffs.append(latitude);
diffs.append(",");
diffs.append(longitude);
diffs.append("\n");

// update found
String name = DatexIINameAnalyzer.GPS2Name(longitude, latitude,
   1);
namesList.add(name);

}
} else {
// for the first time send all retrieved names
String name = DatexIINameAnalyzer.GPS2Name(longitude, latitude,
   1);
namesList.add(name);

}
}

// flush utils on file
FileUtils.printOnFile(ids.toString(), idsF);
FileUtils.printOnFile(diffs.toString(), diffsF);

} else if (table.endsWith("GetSituation")) {

// need to set id for every element
if (oldDoc != null) {
NodeList nodeList = oldDoc.getElementsByTagName("situation");
for (int j = 0; j < nodeList.getLength(); j++) {
Element node = (Element) nodeList.item(j);
// need to set id as ID
node.setIdAttribute("id", true);
}
}

// process
ArrayList<Element> recordList = XMLUtils.extractElementsByTagName
   (newDoc.getDocumentElement(),
"situation");
```

```
// stats
nRecord.set(recordList.size());

for (Element el1 : recordList) {

// extract geo-tagged info
String id  = el1.getAttribute("id");

float longitude = new Float(XMLUtils.extractElementsByTagName(el1
   , "longitude")
.get(0).getTextContent());
float latitude = new Float(XMLUtils.extractElementsByTagName(el1,
    "latitude")
.get(0).getTextContent());

// flush node to file
FileUtils.printOnFile(XMLUtils.toString(el1.cloneNode(true)),
new File(workingDir + "/" + id));

// update ids
ids.append(id);
ids.append(",");
ids.append(latitude);
ids.append(",");
ids.append(longitude);
ids.append("\n");

if (oldDoc != null) {
//System.out.println("Calculate diff of " + table + " for id: " +
    id);
Element el2 = oldDoc.getElementById(id);

if (el2 == null ||
!XMLUtils.toString(el1.cloneNode(true))
.equals(XMLUtils.toString(el2.cloneNode(true)))) {
// update diffs
diffs.append(id);
diffs.append(",");
diffs.append(latitude);
diffs.append(",");
diffs.append(longitude);
diffs.append("\n");

// update found
String name = DatexIINameAnalyzer.GPS2Name(longitude, latitude,
   1);
namesList.add(name);
}
} else {
// for the first time send all retrieved names
String name = DatexIINameAnalyzer.GPS2Name(longitude, latitude,
   1);
namesList.add(name);
}
}

// flush utils on file
FileUtils.printOnFile(ids.toString(), idsF);
FileUtils.printOnFile(diffs.toString(), diffsF);

} else if (table.endsWith("GetMeasuredWeatherData")) {

// extract custom tag
FileUtils.printOnFile(XMLUtils.toString(XMLUtils
.extractElementsByTagName(newDoc.getDocumentElement(), "
   measurementSiteTableReference").get(0)),
new File(workingDir + "/measurementSiteTableReference"));
```

```
FileUtils.printOnFile(XMLUtils.toString(XMLUtils
.extractElementsByTagName(newDoc.getDocumentElement(), "
   headerInformation").get(0)),
new File(workingDir + "/headerInformation"));

// get weather site table
File[] listOfFiles;
boolean found = false;
int i = 0;
do {
listOfFiles = new File("DatexII/npra/
   GetMeasurementWeatherSiteTable").listFiles();
Arrays.sort(listOfFiles);

i = 0;
for (File file : listOfFiles) {
if (file.isFile() & file.getName().endsWith(".xml")) {
found = true;
break;
}
i++;
}
Thread.sleep(50);
} while (!found);

Document weatherSite = XMLUtils.createDOM(FileUtils
.readFromFile(listOfFiles[i])
.toString());

// need to set id for every element
NodeList nodeListSite = weatherSite.getElementsByTagName("
   measurementSiteRecord");
for (int j = 0; j < nodeListSite.getLength(); j++) {
```

```
Element site = (Element) nodeListSite.item(j);
// need to set id as ID
site.setIdAttribute("id", true);
}

if (oldDoc != null) {
NodeList nodeList  = oldDoc.getElementsByTagName("
   measurementSiteReference");
for (int j = 0; j < nodeList.getLength(); j++) {
Element node = (Element) nodeList.item(j);
// need to set id as ID
node.setIdAttribute("id", true);
}
}

// process
ArrayList<Element> recordList = XMLUtils.extractElementsByTagName
   (newDoc.getDocumentElement(),
"siteMeasurements");

// stats
nRecord.set(recordList.size());

for (Element el1 : recordList) {
// extract element containing id
Element el = XMLUtils.extractElementsByTagName(el1, "
   measurementSiteReference").get(0);

// extract geo-tagged info
String id  = el.getAttribute("id");
Element geoEl = weatherSite.getElementById(id);

if (geoEl != null) {
```

```
float longitude = new Float(XMLUtils.extractElementsByTagName(
   geoEl, "longitude")
.get(0).getTextContent());
float latitude = new Float(XMLUtils.extractElementsByTagName(
   geoEl, "latitude")
.get(0).getTextContent());

// flush node to file
FileUtils.printOnFile(XMLUtils.toString(el1.cloneNode(true)),
new File(workingDir + "/" + id));

// update ids
ids.append(id);
ids.append(",");
ids.append(latitude);
ids.append(",");
ids.append(longitude);
ids.append("\n");

if (oldDoc != null) {
//System.out.println("Calculate diff of " + table + " for id: " +
    id);
Element el2 = (Element) oldDoc.getElementById(id).getParentNode()
   ;// this shit of parsing differs from table to table -.-

if (el2 == null ||
!XMLUtils.toString(el1.cloneNode(true))
.equals(XMLUtils.toString(el2.cloneNode(true)))) {
// update diffs
diffs.append(id);
diffs.append(",");
diffs.append(latitude);
diffs.append(",");
diffs.append(longitude);
diffs.append("\n");

// update found
String name = DatexIINameAnalyzer.GPS2Name(longitude, latitude,
   1);
namesList.add(name);
}
} else {
// for the first time send all retrieved names
String name = DatexIINameAnalyzer.GPS2Name(longitude, latitude,
   1);
namesList.add(name);
}
} else {
// no element with this id
}
}

// flush utils on file
FileUtils.printOnFile(ids.toString(), idsF);
FileUtils.printOnFile(diffs.toString(), diffsF);

} else if (table.endsWith("GetMeasurementWeatherSiteTable")) {

// extract custom tag
FileUtils.printOnFile(XMLUtils.toString(XMLUtils
.extractElementsByTagName(newDoc.getDocumentElement(), "
   headerInformation").get(0)),
new File(workingDir + "/headerInformation"));

String[] measurementSiteTable = XMLUtils.toString(XMLUtils
.extractElementsByTagName(newDoc.getDocumentElement(), "
   measurementSiteTable")
.get(0)
```

```
.cloneNode(false)).replace("><", ">\n<").split("\n");
FileUtils.printOnFile(measurementSiteTable[0],
new File(workingDir + "/measurementSiteTableInit"));
FileUtils.printOnFile(measurementSiteTable[1],
new File(workingDir + "/measurementSiteTableEnd"));

FileUtils.printOnFile(XMLUtils.toString(XMLUtils
.extractElementsByTagName(newDoc.getDocumentElement(), "
    measurementSiteTableIdentification").get(0)),
new File(workingDir + "/measurementSiteTableIdentification"));

// need to set id for every element
if (oldDoc != null) {
NodeList nodeList = oldDoc.getElementsByTagName("
    measurementSiteRecord");
for (int j = 0; j < nodeList.getLength(); j++) {
Element node = (Element) nodeList.item(j);
// need to set id as ID
node.setIdAttribute("id", true);
}
}

// process
ArrayList<Element> recordList = XMLUtils.extractElementsByTagName
    (newDoc.getDocumentElement(),
"measurementSiteRecord");

// stats
nRecord.set(recordList.size());

for (Element el1 : recordList) {

// extract geo-tagged info
String id  = el1.getAttribute("id");
```

```
float longitude = new Float(XMLUtils.extractElementsByTagName(el1
   , "longitude")
.get(0).getTextContent());
float latitude = new Float(XMLUtils.extractElementsByTagName(el1,
    "latitude")
.get(0).getTextContent());

// flush node to file
FileUtils.printOnFile(XMLUtils.toString(el1.cloneNode(true)),
new File(workingDir + "/" + id));

// update ids
ids.append(id);
ids.append(",");
ids.append(latitude);
ids.append(",");
ids.append(longitude);
ids.append("\n");

if (oldDoc != null) {
//System.out.println("Calculate diff of " + table + " for id: " +
    id);
Element el2 = oldDoc.getElementById(id);

if (el2 == null ||
!XMLUtils.toString(el1.cloneNode(true))
.equals(XMLUtils.toString(el2.cloneNode(true)))) {
// update diffs
diffs.append(id);
diffs.append(",");
diffs.append(latitude);
diffs.append(",");
diffs.append(longitude);
diffs.append("\n");

// update found
String name = DatexIINameAnalyzer.GPS2Name(longitude, latitude,
   1);
namesList.add(name);
}
} else {
// for the first time send all retrieved names
String name = DatexIINameAnalyzer.GPS2Name(longitude, latitude,
   1);
namesList.add(name);
}
}

// flush utils on file
FileUtils.printOnFile(ids.toString(), idsF);
FileUtils.printOnFile(diffs.toString(), diffsF);

// flush stats
FileUtils.printOnFile(nRecord + ";" + nUpdate + ";", stat, true);

}

// return name list
return namesList;
}



public static String GPS2Name(float longitude, float latitude,
   int step) {
StringBuilder name = new StringBuilder();
int intLon = (int) longitude;
int intLat = (int) latitude;
```

```
name.append("/");
name.append(String.format("%02d", intLon));
name.append("/");
name.append(String.format("%02d", intLat));

longitude = (longitude - intLon)*10;
latitude = (latitude - intLat)*10;

while (step > 0) {
intLon = (int) longitude;
intLat = (int) latitude;
name.append("/");
name.append(String.format("%01d", intLon));
name.append(String.format("%01d", intLat));
longitude = (longitude - intLon)*10;
latitude = (latitude - intLat)*10;
step--;
}

return name.toString();
}

public static float name2GPS(String name, String type) throws
   Exception {
Matcher geo_tag_mtc = geo_tag_rgx.matcher(name);
float point  = 0;
int  index  = 0;
int  offset  = 0;

switch (type) {
case "longitude":
offset = 0;
break;
case "latitude":
```

```java
offset = 1;
break;
default:
throw new Exception("Invalid GPS type");
}
while (geo_tag_mtc.find()) {

// get geo info
String geo_tag = geo_tag_mtc.group();

// convert string into gps point
if (index == 0 & "longitude".equals(type)) {
point = point + new Float(geo_tag);
index++;
} else if (index == 1 & "latitude".equals(type)) {
point = point + new Float(geo_tag);
index++;
} else if (index > 1) {
point = point
+ new Float(geo_tag.substring(0 + offset, 1 + offset))
/ new Float(Math.pow(10, index - 1));
index++;
} else {
index++;
}
}
return point;
}


public static String checkSquare(float longitude,
float latitude,
String workingDir,
File idList) throws IOException {
StringBuilder buf = new StringBuilder();
```

```
// read diff
String diffs = FileUtils.readFromFile(idList).toString();

Matcher diff_mtc = diff_rgx.matcher(diffs);
while (diff_mtc.find()) {
String diff_id   = diff_mtc.group(1);
Float diff_latitude = new Float(diff_mtc.group(2));
Float diff_longitude = new Float(diff_mtc.group(3));

if (diff_latitude >= latitude && diff_latitude <= (latitude +
   side) &&
diff_longitude >= longitude && diff_longitude <= (longitude +
   side)) {
// the record is in the square
buf.append(FileUtils.readFromFile(new File(workingDir
+ "/"
+ diff_id)));
} else {

}
}
return buf.toString();
}

public static String createPublication(String nameTable, String
   content, String baseDir)
throws IOException, SAXException, ParserConfigurationException,
   TransformerException {
switch (nameTable) {
case "GetCCTVSiteTable":
return DatexIINameAnalyzer.createCCTV(content, baseDir);
case "GetSituation":
return DatexIINameAnalyzer.createSituation(content, baseDir);
case "GetMeasuredWeatherData":
return DatexIINameAnalyzer.createWeatherData(content, baseDir);
case "GetMeasurementWeatherSiteTable":
return DatexIINameAnalyzer.createWeatherSiteTable(content,
    baseDir);
default:
return null;

}
}

private static String createWeatherSiteTable(String content,
    String workingDir)
throws IOException, SAXException, ParserConfigurationException,
    TransformerException {
StringBuilder buf = new StringBuilder();
buf.append("<?xml version=\"1.0\" encoding=\"UTF-8\" standalone
    =\"no\"?>");
buf.append(FileUtils.readFromFile(new File(workingDir + "/
    d2LogicalModelInit")));
buf.append(FileUtils.readFromFile(new File(workingDir + "/
    exchange")));
buf.append(FileUtils.readFromFile(new File(workingDir + "/
    payloadPublicationInit")));
buf.append(FileUtils.readFromFile(new File(workingDir + "/
    publicationTime")));
buf.append(FileUtils.readFromFile(new File(workingDir + "/
    publicationCreator")));
buf.append(FileUtils.readFromFile(new File(workingDir + "/
    headerInformation")));
buf.append(FileUtils.readFromFile(new File(workingDir + "/
    measurementSiteTableInit")));
buf.append(FileUtils.readFromFile(new File(workingDir + "/
    measurementSiteTableIdentification")));
```

```
buf.append(content);
buf.append(FileUtils.readFromFile(new File(workingDir + "/
   measurementSiteTableEnd")));
buf.append(FileUtils.readFromFile(new File(workingDir + "/
   payloadPublicationEnd")));
buf.append(FileUtils.readFromFile(new File(workingDir + "/
   d2LogicalModelEnd")));

// create new DOM
Document doc = XMLUtils.createDOM(buf.toString());
doc.normalize();
String table = XMLUtils.toString(doc);

return table;
}

private static String createWeatherData(String content, String
   workingDir)
throws IOException, SAXException, ParserConfigurationException,
   TransformerException {
StringBuilder buf = new StringBuilder();
buf.append("<?xml version=\"1.0\" encoding=\"UTF-8\" standalone
   =\"no\"?>");
buf.append(FileUtils.readFromFile(new File(workingDir + "/
   d2LogicalModelInit")));
buf.append(FileUtils.readFromFile(new File(workingDir + "/
   exchange")));
buf.append(FileUtils.readFromFile(new File(workingDir + "/
   payloadPublicationInit")));
buf.append(FileUtils.readFromFile(new File(workingDir + "/
   publicationTime")));
buf.append(FileUtils.readFromFile(new File(workingDir + "/
   publicationCreator")));
```

```
buf.append(FileUtils.readFromFile(new File(workingDir + "/
   measurementSiteTableReference")));
buf.append(FileUtils.readFromFile(new File(workingDir + "/
   headerInformation")));
buf.append(content);
buf.append(FileUtils.readFromFile(new File(workingDir + "/
   payloadPublicationEnd")));
buf.append(FileUtils.readFromFile(new File(workingDir + "/
   d2LogicalModelEnd")));

// create new DOM
Document doc = XMLUtils.createDOM(buf.toString());
doc.normalize();
String table = XMLUtils.toString(doc);

return table;
}

private static String createSituation(String content, String
   workingDir)
throws IOException, TransformerException, SAXException,
   ParserConfigurationException {
StringBuilder buf = new StringBuilder();
buf.append("<?xml version=\"1.0\" encoding=\"UTF-8\" standalone
   =\"no\"?>");
buf.append(FileUtils.readFromFile(new File(workingDir + "/
   d2LogicalModelInit")));
buf.append(FileUtils.readFromFile(new File(workingDir + "/
   exchange")));
buf.append(FileUtils.readFromFile(new File(workingDir + "/
   payloadPublicationInit")));
buf.append(FileUtils.readFromFile(new File(workingDir + "/
   publicationTime")));
```

```
buf.append(FileUtils.readFromFile(new File(workingDir + "/
   publicationCreator")));
buf.append(content);
buf.append(FileUtils.readFromFile(new File(workingDir + "/
   payloadPublicationEnd")));
buf.append(FileUtils.readFromFile(new File(workingDir + "/
   d2LogicalModelEnd")));

// create new DOM
Document doc = XMLUtils.createDOM(buf.toString());
doc.normalize();
String table = XMLUtils.toString(doc);

return table;
}

private static String createCCTV(String content, String
   workingDir)
throws IOException, SAXException, ParserConfigurationException,
   TransformerException {
StringBuilder buf = new StringBuilder();
buf.append("<?xml version=\"1.0\" encoding=\"UTF-8\" standalone
   =\"no\"?>");
buf.append(FileUtils.readFromFile(new File(workingDir + "/
   d2LogicalModelInit")));
buf.append(FileUtils.readFromFile(new File(workingDir + "/
   exchange")));
buf.append(FileUtils.readFromFile(new File(workingDir + "/
   payloadPublicationInit")));
buf.append(FileUtils.readFromFile(new File(workingDir + "/
   publicationTime")));
buf.append(FileUtils.readFromFile(new File(workingDir + "/
   publicationCreator")));
```

```
buf.append(FileUtils.readFromFile(new File(workingDir + "/
    genericPublicationName")));
buf.append(FileUtils.readFromFile(new File(workingDir + "/
    genericPublicationExtensionInit")));
buf.append(FileUtils.readFromFile(new File(workingDir + "/
    cctvSiteTablePublicationInit")));
buf.append(FileUtils.readFromFile(new File(workingDir + "/
    headerInformation")));
buf.append(FileUtils.readFromFile(new File(workingDir + "/
    cctvCameraListInit")));
buf.append(FileUtils.readFromFile(new File(workingDir + "/
    cctvCameraListVersionTime")));
buf.append(content);
buf.append(FileUtils.readFromFile(new File(workingDir + "/
    cctvCameraListEnd")));
buf.append(FileUtils.readFromFile(new File(workingDir + "/
    cctvSiteTablePublicationEnd")));
buf.append(FileUtils.readFromFile(new File(workingDir + "/
    genericPublicationExtensionEnd")));
buf.append(FileUtils.readFromFile(new File(workingDir + "/
    payloadPublicationEnd")));
buf.append(FileUtils.readFromFile(new File(workingDir + "/
    d2LogicalModelEnd")));

// create new DOM
Document doc = XMLUtils.createDOM(buf.toString());
doc.normalize();
String table = XMLUtils.toString(doc);

return table;
}


}
```