The Anyplace 4.0 IoT Localization Architecture

Paschalis Mpeis*, Thierry Roussel†, Manish Kumar+, Constantinos Costa‡, Christos Laoudias*, Denis Capot-Ray†, Demetrios Zeinalipour-Yazti*

*University of Cyprus, P.O. Box 20537, 1678 Nicosia, Cyprus
† Alstom SA, 93400 Saint-Ouen, France
+ Infosys Ltd, 560100 Bangalore, India
‡ University of Pittsburgh, PA 15260, USA
pmpeis01@cs.ucy.ac.cy, thierry.roussel@alstomgroup.com, manish.kumar156@infosys.com, costa.c@cs.pitt.edu, laoudias@ucy.ac.cy, denis.capot-rey@alstomgroup.com, dzeina@cs.ucy.ac.cy

***Abstract*—The Internet of Things (IoT) revolution has massively introduced sensor-rich tracking devices to an ever growing landscape of smart spaces (e.g., factories, hospitals, and ships). One problem that remains unsolved over the years is the localization problem for IoT, given that Satellite-based solutions are inaccurate in indoor spaces where human activity takes place 80-90% of the time. In this paper, we introduce a novel open-source architecture for IoT localization, coined *Anyplace 4.0 IoT (A4IoT)*, which exploits *signal fingerprinting* to organize under the same roof a wide range of different localization technologies (e.g., Wi-Fi, BLE, Cellular, UWB, Computer Vision). We present the technical requirements of A4IoT inspired by the Alstom SA smart factory, operating worldwide in rail transport markets. A4IoT comprises a crowdsourcing architecture where deployers can collect and organize fingerprint signals inside smart spaces in a designated localization service running on the Edge (from Raspberry to Datacenter). The service incorporates timeseries databases for tracking targets and deployers can provide accurate room-level localization accuracy ($\approx$ 2 m) on a variety of platforms (e.g., Android, Linux, Mac, Windows, Robot OS) but also integrate A4IoT through Web 2.0 endpoints to their software ecosystems.**

***Index Terms*—IoT, mobile, indoor, location, crowdsourcing**

I. Introduction

Internet of Things (IoT) refers to a large number of physical devices being connected to the Internet that are able to see, hear, think, perform tasks as well as communicate with each other using open protocols [1]–[4]. IoT devices are connected to Cloud and Edge computing appliances through massively parallel I/O channels (e.g., 5G, WiFi6) with millisecond latency offering new opportunities in industrial optimization, human health and well-being as well as safety. A typical household in the developed world currently operates around 5 to 10 smart devices, and this number is expected to increase to 500 [5] by 2022. TVs, power plugs, light bulbs, security sensors to name a few, become interconnected with open protocols [6]–[9], creating whole new ecosystems. In absolute numbers, the IoT revolution is expected to bring the number of such devices close to a staggering 40 billion in 2020, more than double from 2019 [10].

The omni-present availability of sensor-rich smartphones and IoT devices along with the fact that people spend 80-90% of their time in indoor environments has boosted an interest around indoor location-based services, such as in-building guidance and navigation, inventory management, marketing and elderly support through Ambient and Assisted Living [11], [12]. The key enablers for the uptake of such indoor applications are what nowadays we call the open-source *Localization Services*, like Anyplace [13], Redpin.org [14] and Find3 [15]. These consist of indoor models, such as floormaps and *Points-of-Interest (POIs)*, along with wireless, light and magnetic signals used to localize users. Anyplace 3.4[1] [16], has been a predecessor technology for A4IoT providing infrastructure-free 3-D (dimensional, i.e., floor-level) localization combined with modeling and crowdsourcing under the same hood. Even though the open source software stacks have helped to the wider adoption of important scientific findings (enabling the integration of new localization algorithms) but also transparency of what happens behind the service, none of these architectures provide elements for IoT tracking at Internet-scale (e.g., Cloud) and at Minuscale (e.g., Raspberry).

In this paper, we introduce a novel open-source architecture for IoT localization (see Fig. 1), coined *Anyplace 4.0 IoT (A4IoT)*, which exploits *signal fingerprinting* to organize under the same roof a wide range of different localization technologies (e.g., Wi-Fi, BLE, Cellular, UWB, Computer Vision). We present the technical requirements of A4IoT inspired by the

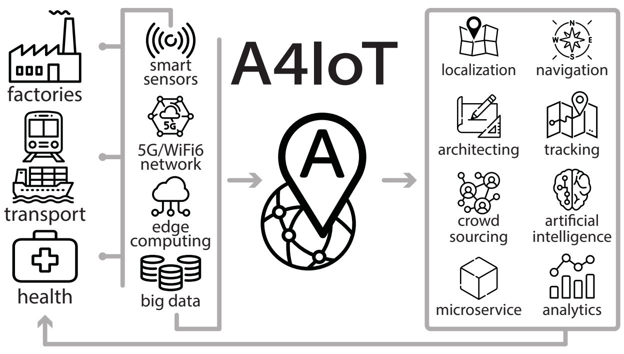


Fig. 1: *A4IoT* is a localization service for the Edge aligned towards the needs of the new industrial IoT revolution in various smart spaces (e.g., factories, health and ships).

[1]Anyplace. `https://anyplace.cs.ucy.ac.cy/`

Fig. 2: Three real life scenarios have motivated the A4IoT architecture. **(Left)** Alstom deploys our architecture to built a smart factory (main focus of this paper). **(Center)** Lash-Fire will deploy our indoor localization for emergency response of fire fighters on cargo ships. **(Right)** Endorse will navigate its robotic fleet to provide elderly care to the aging population.

Alstom SA smart factory, operating worldwide in rail transport markets. A4IoT comprises a crowdsourcing architecture where deployers can collect and organize localization signals inside smart spaces in a designated localization service running on the Edge (from Raspberry to Datacenter). The service incorporates timeseries databases for tracking targets and deployers can provide accurate room-level localization accuracy (≈ 2 m) on a variety of platforms (e.g., Android, Linux, Mac, Windows, Robot OS) but also integrate A4IoT through Web 2.0 endpoints to their proprietary software ecosystems.

The localization literature is very broad and diverse as it exploits several technologies. GPS is obviously ubiquitously available but has an expensive energy tag and is also negatively affected by the environment (e.g., cloudy days, forests, downtown areas). Besides GPS, the localization community proposed numerous proprietary solutions including: *Infrared, Bluetooth, visual or acoustic analysis, RFID, Inertial Measurement Units, Ultra-Wide-Band, Sensor Networks, Wireless LANs, etc.*; including their combinations into hybrid systems [17]. A4IoT uses a hybrid Radiomap-based indoor localization, which stores radio signals from (like Wi-Fi APs, BLE/UWB/Cellular/CV) in a database at a high density.

We next explain the localization subsystem of A4IoT, which utilizes the notion of fingerprinting in order to bring forward a localization technology that works seamlessly on the edge/IoT device with optional extensions for intermittent connectivity [18] and privacy [19]. The A4IoT localization process works as follows: in an offline phase a logging application records the so called *fingerprints*, which consist of Received Signal Strength Indicator (RSSI) of these sensors at certain coordinates (x,y) pin-pointed on a building floor map (e.g., every few meters). Subsequently, in a second offline phase the sensor fingerprints are joint into several NxM matrices, coined the *RM* (i.e., one *RM* per sensor type), where N is the number of unique (x,y) fingerprints and M the total number of beacons. Finally, a user or IoT device can compare its currently observed fingerprint against the respective sensor-type *RM* in order to find the best match, using known algorithms such as KNN or WKNN [20]. Our solution is infrastructure-free, as it operates best effort with whatever sensor-type is available.

It is important to notice that the requirement of localization accuracy is very use-case dependent. For example, 50cm is required for realtime guidance, 2m is required for asset localization like car, forklift and any big parts. While ultra-fine accuracy is still a challenge with many technologies (like Wi-Fi), current achievable accuracy is acceptable for many use-cases. Our goal is to integrate different types of *RM* to make 3-D localization available in a variety of environments. A4IoT is designed to cope with the Industry 4.0 and motivated by the following three real-world scenarios:

Asset tracking at Alstom SA[2]: This scenario, which is detailed extensively throughout the paper, incorporates A4IoT in a rail manufacturer corporation to add location-awareness for tracking factory assets (see Fig. 2, left). The manufacturer aims to optimize routes, enhance guidance, improve safety and supply reliability, and reduce warehouse overall costs. The system must easily integrate to a complex software ecosystem of services with the predominant role of a containerized *Localization Service* using existing Wi-Fi technology, while visualization tools should provide analytics for internal reports and monitoring. Support for new localization technologies is also considered, such as Ultra-Wideband (UWB) (available in iPhones 11 [21]) or LoRa [22], with the aim of improving accuracy.

Emergency response with Lash-Fire[3]: This scenario incorporates A4IoT to increase fire safety on *RO-RO* (roll on, roll off) cargo vessels that transport cars. What is necessary is a backend on the edge (i.e., the vessel itself), capable of providing 3D localization means with room-level accuracy. The A4IoT must integrate as a localization service next to the Ship Information System (see example from the Stena Jutlandica Ro-Ro/Passenger Ship in Fig. 2, center) enabling to track spatio-temporal targets in real time, to detect fire hazards and guide fire-personnel. In the lack of infrastructure, this scenario focuses on no-infrastructure localization with computer vision fingerprinting techniques (e.g., using YOLO or Google ARCore).

Robotic fleet in healthcare with Endorse[4]: This scenario aims of creating a safe, efficient, and integrated fleet of robots for logistic applications in health-care (see Fig. 2, right).

[2]Alstom: `https://www.alstom.com/`
[3]Lash-Fire: `https://lashfire.eu/`
[4]Endorse: `https://www.endorse-project.eu/`

The deployment of A4IoT on custom environments like the RobotOS [23] is necessary, for carrying equipment, drugs, and give assistance to elderly people. It will additionally track personnel, patients, and equipment. Given that Robots have a variety of sensors for local IMU-based localization, A4IoT is the component bringing the global localization perspective.

The contributions of this work are summarized as follows:

- We propose a new containerized backend architecture, able to run from a low-end configurations to powerful multi-node clusters with minimal effort;
- We propose a data layer that can accommodate new signals for localization, handle high volumes of spatio-temporal data and present visual analytics in realtime;
- We provide a modular library that allows the deployment of clients in no-time on a diverse set of frontends.

Section II describes the business requirements of Alstom SA for incorporating Industry 4.0 in their factories. It is followed by Section III that gives an overview of our new backend architecture. Section V describes our modular library that enables easy deployment of A4IoT in any environment. In Section VI we detail the necessary changes to our data layer to support high-velocity spatio-temporal data, while decreasing our minimum hardware requirements. Our realtime visualization analytics are described in Section VII and our concluding remarks are in Section VIII.

II. A4IoT Technical Requirements

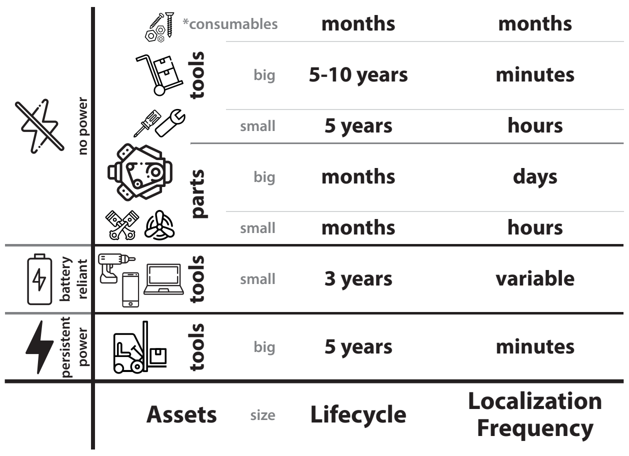
This section showcases how Internet of Things (IoT) requirements are materialized in a real-world business. Particularly, we describe the business requirements of Alstom SA to incorporate localization technologies into their IoT processing stack. We analyze the IoT applications and categorize them based on the asset typology across different departments. We also analyze specific use-cases that define key IoT criteria in asset tracking and elaborate on how Anyplace 4.0 fits their business requirements.

A. IoT Applications per Asset Typology

Assets are divided based on their typology into three categories, namely *workers*, *tools* and *parts*. They are associated with interconnected buildings, which collectively constitute a *smart factory*. Such buildings are offices, depots, and manufacturing plants. All indoor localization use-cases span over different stakeholders within the company. These are the departments of Environment Health and Safety (EHS), Information Technology (IT), Industrial, and Facilities.

Workers: Location data analysis and emergency alerts could improve the worker's safety. For example, when a worker enters a prohibited section of the facility and a hazardous operation is underway. Localization data can also enhance the work guidance especially when specialized vehicles (e.g., forklifts) are involved by optimizing the routing.

Tools: Realtime tracking can significantly improve the tool management. Location context can increase tools utilization and therefore availability. The process of checking-in or



	Assets	size	Lifecycle	Localization Frequency
no power	*consumables		months	months
no power	tools	big	5-10 years	minutes
no power	tools	small	5 years	hours
no power	parts	big	months	days
no power	parts	small	months	hours
battery reliant	tools	small	3 years	variable
persistent power	tools	big	5 years	minutes

Fig. 3: Asset categories based on IoT criteria for tracking.

checking-out a tool can be automated, maximizing prevention on accidental losses or even theft. With data analysis the usage patterns of the tools can be deducted, which will provide insights on their performance checks and maintenance. Finally, a more accurate and automated management of the tools can potentially reduce their fleet, which will decrease overall costs.

Parts: A precise tracking of parts can increase the supply-chain reliability and is divided into three phases. In the first phase, a part is stored in a depot and therefore it is mostly stationary. In the second phase, a part is within a container and in transit. In the last phase, a part is moving within a manufacturer plant and progresses on the assembly line. With realtime tracking, the risk of under-stocking or over-stocking parts will be eliminated. Additionally, data analysis aggregated by time and accurate time tracking for each part in each phase can expose the manufacturing efficiency. All above collectively contribute to an IoT-based smart inventory system that minimizes the warehouse management costs.

B. Key IoT Localization Criteria in Asset Tracking

An extensive study was done to determine the key IoT criteria for asset tracking in *smart factories*. With power source being a major point of consideration, we have divided assets in three categories, as shown in Fig. 3. In the first category assets have persistent power supply. In the second category assets rely on a battery source. In the final category assets have no power supply whatsoever.

Persistent power: In this category assets are assumed to have a constant power supply. A real example for this category is a vehicle asset, for example a *forklift*. In this example, the asset is expensive with a long lifecycle. Also, the size of the asset is large, and the target localization accuracy is 2 meters. Solutions include attaching a Raspberry PI or an Android-powered smartphone on the board of the vehicle. Both solutions will use vehicle's power sources, and report Wi-Fi RSSI values and/or GPS coordinates to the Anyplace server. The smartphone application and the Raspberry PI will use our Android client and command line interface respectively, which are described in Section V.

Battery-reliant: Assets like tools or workers will utilize battery-reliant solutions for localization. The power source is assumed to be adequate for a full working day. Workers in particular will utilize smartphones, tablet devices, or laptops. Those devices will have built-in support for Wi-Fi, GPS, and/or mobile data. Examples of tool assets include cordless tools like drills or wrenches. Those do not support any communication or localization technologies at the hardware level. All battery-reliant assets have a lifecycle of roughly 3 years. The solution for smart devices used by workers is to utilize a modified version of our Android client for communicating with the Anyplace server. For tools, externally attached modules will augment their functionality to incorporate Wi-Fi, GPS, and/or mobile data. Then, custom scripts embedded into the attached modules will communicate with our service. Those scripts will utilize the Anyplace Java library.

No power: Assets like tools or parts may come without a power source. Tools have a lifecycle of many years and are divided according to their size, small or big. Parts, which have a lifecycle of a few months, are also divided into small or big. There is, however, a special third category for consumable parts, like bolts or nuts. Small tools, like hammers or screwdrivers, move quite often and require 0.5-meter accuracy. Some big tools, like trolleys, require constant tracking. For other big tools, like jigs, periodic tracking suffices. In both cases the target accuracy is 2 meters. Small parts usually have low costs, so the tracking mechanism should rely on *passive tags*. Such tags rely on other equipment, i.e., a smartphone, for tracking. Big parts tend to move less than once in a day. Consumables come in large quantities, have a very small footprint, and a low cost per unit thus their tracking is not necessary.

The solution covering all tools and small parts is to utilize *passive tracking*, such as UWB tags. They are small enough to be attached on those assets and will be supported by Anyplace 4.0. Small parts or tools can also utilize other forms of passive tracking, such as barcodes, or RFID tags. The same solution can be applied to track cabinets filled with consumables. As an added functionality, an embedded device with a weight sensor attached to the cabinet can estimate quantities and update inventories for consumables. For infrequently-moving assets, like big tools or particular parts, *active tracking* can be used. Specifically, a low-powered device will be utilized to report a few times a day positioning through Wi-Fi or a mobile network. All described solutions will be using our library to feed Anyplace servers with localization context.

III. A4IoT Architecture

The A4IoT architecture is divided into five main components, visualized in Fig. 4. The *Server* and the *Data Store* constitute the backend, and the *Web*, the *Library*, and the *IoT clients* constitute the front-end. The additions for the A4IoT are emphasized with dotted lines and are described in Sections IV-VII.

Backend: The *Server* component contains the application logic of the A4IoT service. It is implemented using Play [24], a lightweight MVC framework for web applications. A RESTful API is exposed that enables crowdsourcing, spatio-temporal queries, and has interfaces for various data stores. Other features include visual analytics, tiling of architectural floormaps, and secure *OAuth 2.0* authentication. The *Data Store* component is responsible for storing and retrieving raw data using a Distributed Filesystem (DFS), IoT data using a time-series store, and JSON objects using a document store.

Front-end: *Architect*, *Viewer*, and *Analytics* modules are *Web Apps* built with *HTML5*, *CSS3*, and *AngularJS*. *Architect* allows users to insert buildings by uploading architectural floorplans and then interactively designing on top of them routes and Point of Interest (POI) elements. *Viewer* is a search and navigation engine built on top of the crowdsourced data. *Analytics* is a data visualization dashboard that interacts with the *Data Store* and is described in Section VII. *Logger* and *Localization* are native Android applications that enable fingerprinting and localization, respectively [25].

Logger in particular, enables users to crowdsource collected Wi-Fi RSSI values by recording them from nearby Wi-Fi Access Points and batch-feeding them to the *Server*. *Localization* allows users to view their location inside buildings. It also allows navigation between POIs, similarly with *Viewer*. *Localization* however has superb accuracy as it queries the A4IoT backend with collected Wi-Fi RSSI values. Additional sensor input is used too (i.e., accelerometer, compass, and gyroscope) to enhance the navigation experience. For both localization and navigation, the results are drawn on top of architectural floorplans for uploaded buildings, with multiple floors supported.

IV. A4IoT Containerization

This section describes the containerization of the A4IoT software stack. It utilizes Docker to enable the deployment of single-node or multi-node configurations, regardless of any dependencies or the underlying OS.

A. Single-node deployment

The A4IoT docker image initially installs any software dependencies used by the service. Then, depending on the target environment, development or production, relevant code is pulled from A4IoT's GitHub repository using Jenkins and compiled using the *sbt* tool. Configuration for all A4IoT components is conveniently managed through a single *.env* file, removing any sensitive information from the source code. Then, separate container instances are spawned for each database service to enhance security and resource handling. All communication is encrypted even in internal networks, as A4IoT image automatically creates and uses SSL certificates based on the configuration. Also, any relevant database initialization, including bucket and views creation, is automatically handled by the image. Finally, A4IoT is launched as a service and a set of scripts allow operations like automatically trimming the log files or managing the lifecycle of the service.

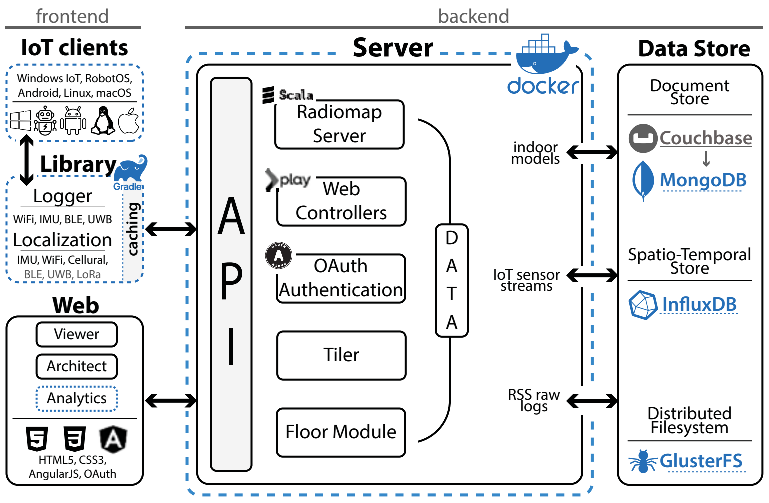


Fig. 4: Architectural overview of A4IoT.

B. Multi-node cluster deployment

In the case of a *cluster deployment*, additional configuration is needed for each node of the cluster that can be passed through the *.env* files. Specifically, it allows plugging in external database or DFS clusters, and SSL certificate authorities. Finally, on a separate node a HaProxy [26] container is spawned and set-up to balance the incoming load and provide protection from attacks like DDoS.

A4IoT Cluster: Our production environment uses a 3-node cluster with two replicas, for both the database engines and the DFS. This setup allows for a full operation with just a single node active at any time and clearly adopters of A4IoT can adjust the configuration scale to their requirements. For the DFS in particular, we configured GlusterFS that exposed exceptional performance for a variety of loads. Our results indicate that for a 2 GB file we can achieve 136 MB/s sequential read and 127 MB/s sequential write, which is satisfactory for our purpose. For much more massive loads the Hadoop HDFS file system, with its 128MB block size and replication factor of 3, provides much more scalable and fault-tolerant alternatives for datacenter scenarios.

For the public running instance of A4IoT we have also reconfigured our network configuration on the hardware, OS virtualization level and security level. Particularly, we have created a Demilitarized Zone (DMZ) network to isolate our service from any other machines in our clusters. In DMZ resides only the *load-balancing node (LB)* that runs the HAProxy container that also thwarts data crawlers. All nodes, including LB, are part of an isolated network. This network is unroutable by others, allowing only outgoing Internet connections to satisfy 3rd-party service dependencies like *OAuth 2.0*, or software vulnerability updates. The network uses 2 physical uplinks, which are configured for fault-tolerance, load-balancing, and double throughput.

V. A4IoT Library and IoT clients

This section describes the front-end libraries that simplify development of A4IoT clients. A list of the currently developed libraries and clients is shown in Table I.

A. Java library: anyplace-core

The *anyplace-core* Gradle library encapsulates the main functionality for IoT clients. It is as generic as possible to facilitate clients' development for different environments and operating systems. The library emits statically or dynamically generated information to the users. Finally, it allows dynamic switching between different A4IoT backends, and wraps each API endpoint with code that transparently handles communication with the backend.

Static Content: It relates mostly to raw, infrequently mutating data, such as buildings. There are endpoints that return buildings to the proximity of a user, buildings that belong to a campus, or the complete list of all the buildings. Given a building, the library can return a list of all the floors and all the POIs. The list of POIs can be further narrowed down to the ones that belong to a particular floor. It may also return the valid connection points between different POIs. Finally, it may return raw, crowdsourced data, which include RadioMap (RM) of Wi-Fi RSSI values, heatmaps of the Wi-Fi coverage, and tiled architectural floorplans.

Dynamic Content: It relates to user localization and tracking requests. There are endpoints that return navigation directions, either between two points or two POIs. It also provides server-side or offline localization on the IoT. The former sends to the server a list of Wi-Fi RSSI values and gets back and an estimated user location. The latter requests from the server a bounding box of pre-generated RM, and then uses it in combination with the Wi-Fi RSSI values to run the localization algorithms on the IoT.

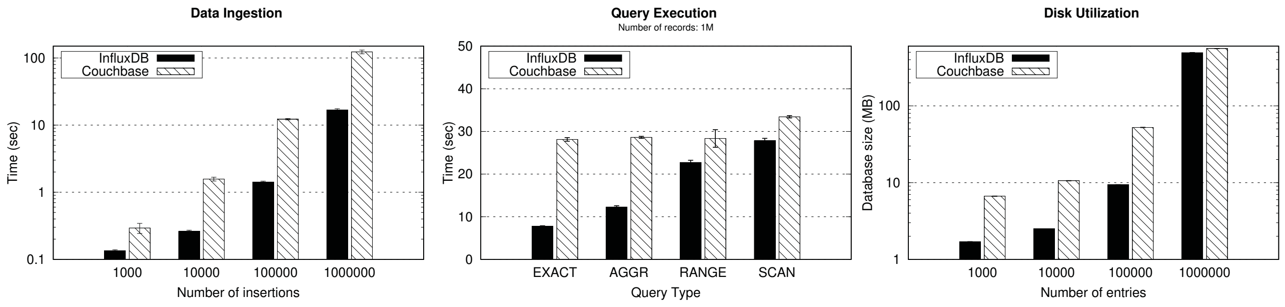


Fig. 5: **(Left)** Time-series vs. document-store comparison for data ingestion. **(Center)** Various query operation types. **(Right)** Disk utilization.

Language	**Type**	**Published**
Java	Library (anyplace-core)	Gradle
OS	**Type**	**Published**
Linux-based	Tool	-
macOS	Tool	-
Windows	Tool	-
Android	Library (anyplace-android)	Gradle
Android	Application	Google Play
ROS	Application	Robot App Store

TABLE I: List of libraries, applications, and tools that are developed in the scope of the A4IoT architecture.

B. Command Line Interface (CLI)

The `anyplace-core` library also has a CLI. Instead of including the library JAR file into another Java-based project, it is executed directly from shell applications from either local cache or the A4IoT service. The snippet in Fig. 1 performs a server-side localization using a *bash script*. The Wi-Fi RSSIs are extracted and fed to the tool along with the user's id, the floor that the user is in, and the localization algorithm. While it works specifically for Linux, with a minor adjustment it can also run on macOS. In particular, the Wi-Fi RSSI scanning should be performed through the Network Manager tool `nmcli`[5] instead. The CLI is particularly convenient as it allows developing IoT clients with minimal effort.

```bash
#!/bin/bash
ALGO=$KNN
interface="wlo1"
# Get RSS values (Linux specific)
rssi=$(jsonFormat $(iwlist $interface scan))
# Localize
CMD=$(java -jar anyplace.jar)
$CMD -estimatePosition $user $floor $algo $rssi
```

Listing 1: Localization on Linux using the A4IoT CLI.

The settings that relate to the backend servers are read from a configuration file that resides on a user's home directory. It is initialized with reasonable defaults and then the user is advised on how to modify it.

C. Android Library and Demo Client

Android provides its own APIs for carrying out tasks for increased security and performance. This additional functionality is included into *anyplace-android*, another Gradle library that includes the *anyplace-core* library.

[5]NetworkManager: `https://linux.die.net/man/1/nmcli`

```java
// load backend settings from user preferences
SharedPreferences sp = getSharedPreferences(..);
String host = sp.getString("pref_host", _HOST);
String port = sp.getString("pref_port", _PORT);
// Get private storage within the Android app
String cache = getApplicationContext().getFilesDir();
// Create a connection & fetch a building's floors
Anyplace ap = new Anyplace(host, port, cache);
String response = ap.allBuildingFloors(b_id);
```

Listing 2: Demo client utilizes the *anyplace-android* library to fetch all floors of a given building id.

The *anyplace-android* library declares any permissions that are necessary for the successful operation of the service. These include Internet access, permissions to read the Wi-Fi RSSI values, ability to run code in the background, and access to storage. Network communications and subsequent computations are wrapped into asynchronous tasks as necessary, since their associated overheads might affect the UI responsiveness. Finally, offline navigation data are securely cached and managed by a client application in private storage.

The *anyplace-android* library can then be used to create clients, like the demo application shown in Listing 2. The client has a dedicated user-preferences XML file for the backend configuration, which enables dynamic switching to different servers. This enables individuals to use local, privacy-preserving deployments to manage their *smart-homes*, and effortlessly switch to bigger clusters when needed like ours, which has accumulated data from tens of thousands of users.

VI. A4IoT Spatio-Temporal Data Store

This section describes the additions to the *Data Store* component of A4IoT. A spatio-temporal database is integrated into the backend to efficiently consume input streams of IoT sensor data. Additionally, existing views are being migrated to MongoDB [27] in an effort to reduce the minimum hardware requirements.

Time-Series store: For the efficient consumption of raw IoT data a *time-series* database is incorporated into the *Data Store*. This enables real-time analytics and visualizations presented in Section VII, instead of selectively performing them with manual batch operations. In particular, we have implemented controllers for InfluxDB [28], which was consistently ranked top for write-intensive workloads. It is also optimized for spatio-temporal queries, in contrast with CouchbaseDB [29]

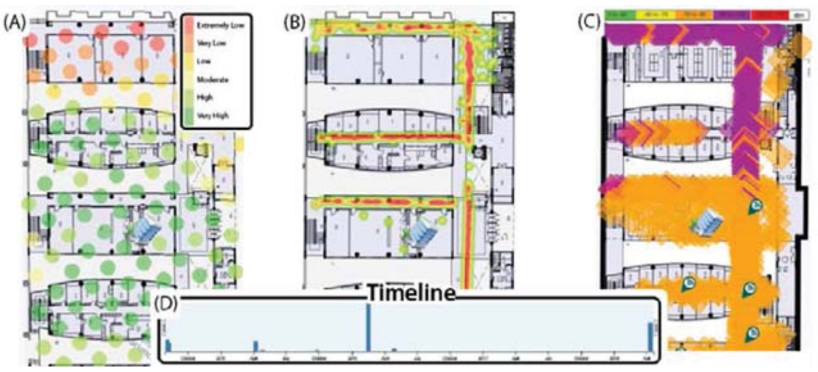


Fig. 6: Fingerprint Management Studio (FMS) provides analytics that visualize: (A) accuracy estimation using Wi-Fi readings; (B) the density of collected Wi-Fi fingerprints, (C) the Wi-Fi coverage; and (D) the Timeline can filter results on particular periods in time.

that is a more generic *document-store*. With temporal queries we can now consume high-volume input streams to provide real-time IoT tracking. For spatial queries we are gradually replacing our existing spatial views that are inefficient in CouchbaseDB. In particular, we are replacing the views responsible for processing data that generate heatmaps and RM files, and for the server-side localization. We have found that those views get invalidated quite frequently, spiking high disk usage in our clusters.

To track IoT devices somebody could complementary also have used complete IoT tracking platforms, like Thingsboard.io (for general IoT) or OpenHAB (for smart homes). In the future we intend to integrate A4IoT into these ecosystems in the form of widgets (i.e., loading A4IoT assets through the GUIs of these instruments). Providing native tracking capabilities from within A4IoT, as opposed to third party tools, is important to simplify the anatomy of the A4IoT *Service Oriented Architecture (SOA)*.

Additionally, we implemented two endpoints that combine spatial and temporal semantics. The first, retrieves entries in a timespan that are within two geometric points. The second, retrieves points within a *geometric fence (geofence)* that were submitted for a particular period in time. The geofence is essentially a variable-length radius applied to a particular geometric point. Those are used for our Fingerprint Management Studio (FMS) module, described in Section VII.

Document-store: The initial version of our software used CouchbaseDB, which is a document store with a built-in distributed in-memory layer (memcached), enabling easy eventually consistent distributed replication. However, the computational requirements of this approach constitutes this inefficient for edge devices, such as, the popular *ARMv7* Raspberry PI. As such, *A4IoT* is expected to migrate to the less demanding MongoDB document store in the near future.

Micro-benchmarking: In Fig. 5 we present the results of our *Data Store* evaluation by comparing InfluxDB against CouchbaseDB. We have used datasets ranging from 1 thousand to 1 million entries, built from real RM data from Anyplace. Error bars show the standard deviation and results are presented on a logarithmic scale, where appropriate.

Fig. 5 (left) shows data ingestion with different number of insertions. We observe that InfluxDB has almost an order of magnitude less ingestion time for any number of insertions. For 1 million entries, InfluxDB is impressively 7x faster than CouchbaseDB. Fig. 5 (center) shows the performance of InfluxDB against CouchbaseDB over a variety of query operations. The time-series database is consistently faster in all cases. Particularly, for the *RANGE* query, which filters out about 50% of the tuples over a time-slice, InfluxDB is 28% faster. InfluxDB is 2.3x faster considering the *AGGR* query that aggregates the output of the *RANGE* query. For the *EXACT* query, which returns a random entry from the database, InfluxDB is 3.6x faster. Finally, the *SCAN* query simply emits all the entries and still the InfluxDB is 20% faster. Fig. 5 (right) shows that InfluxDB requires from 13% to 5.6 times less space than CouchbaseDB.

VII. A4IoT Analytics & Visualization

This section presents additions to the *Analytics* component. By raw IoT data analysis we are able to deduce valuable information over accumulated data. FMS, shown in Fig. 6, is a signal management studio that is fully integrated into A4IoT. It manages a collection of location dependent sensor readings for indoor environments and provides visual analytics. The implemented visual operators into A4IoT are:

Accuracy Estimation: It is a visual accuracy assessment tool for localization at arbitrary locations. Our approach applies a black-box technique for fingerprint interpolation based on *Gaussian Processes*. It allows us to predict sensor readings at chosen locations given the initial *RM*, and then estimate the uncertainty of such predictions. Finally, we can derive a theoretically, solid lower bound for the uncertainty in the location estimation, i.e., the localization error, in the form of a *Cramer-Rao Lower Bound* (CRLB). We utilize the derived CRLB as the *localization score* and show its results in the form of a heatmap as shown in Fig. 6 (A).

Fingerprint heatmaps: It visualizes the density of the collected Wi-Fi fingerprints that was done while mapping a building, as shown in Fig. 6 (B). With red color the density is higher. It can be useful to the crowdsourcing community as it shows which areas need denser fingerprints.

Wi-Fi Coverage: It is an estimate of the Wi-Fi coverage that is anticipated within buildings, as shown in Fig. 6 (C). It also shows the expected data rates. It can easily show the areas that need more Wi-Fi Access Points to be installed, in order to improve the network coverage and/or speeds.

For all visual analytics a temporal window can be set, shown in Fig. 6 (D). These filters result to a time-slice in the past, or to new incoming data. With the enhancements presented in SectionVI, a time-series database accomplishes those tasks in real-time, replacing our previous approach where select buildings were analyzed in batch and subsequently cached.

VIII. Conclusions and Future Work

In this paper, we introduce a novel open-source architecture for IoT localization, coined *Anyplace 4.0 IoT (A4IoT)*, which exploits *Fingerprinting* principles to organize under the same roof a wide range of different localization technologies (e.g., Wi-Fi, BLE, Cellular, UWB and CV). We present the technical requirements of A4IoT inspired by the Alstom SA smart factory, operating worldwide in rail transport markets. We have implemented an InfluxDB controller, to consume high-volumes of IoT generated data, and we are gradually migrating the remaining of our data views to the more lightweight MongoDB. This allows our architecture to run on low-end single board computers, placed on the edge, while being able to scale to sophisticated multi-node cluster setups. In the future we aim to also investigate the usage of higher location accuracy technologies, community integration blockchain, and Web 3.0. We will also investigate map-matching algorithms to localize the RM on the backend graph and CV localization.

Acknowledgements: We would like to thank the researchers at the Data Management Systems Laboratory of the University of Cyprus and the Anyplace Community at wide that have contributed to this open-source project over the years. The A4IoT extensions are supported in part by Cyprus' Research Promotion Foundation RESTART 2016-2020 programme, under project EnterCY INTEGRATED/0609/0020; EU's Horizon 2020 research and innovation programme project LASH FIRE under grant agreement nº 814975; and EU's Marie Skłodowska-Curie Actions (MSCA) H2020-MSCA-RISE-2018 project ENDORSE, under grant agreement nº 823887. The fifth author is supported by EU's Horizon 2020 research and innovation programme under grant agreement nº 739551 (KIOS CoE) and from the Republic of Cyprus through the Directorate General for European Programmes, Coordination and Development.

References

[1] L. Yao, Q. Z. Sheng, and S. Dustdar, "Web-based management of the internet of things," *in IEEE Internet Computing, vol. 19, iss. 4, pp. 60–67*, 2015.

[2] A. A. Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Comm. Surv. Tutor., vol. 17, no. 4, pp. 2347–2376*, 2015.

[3] S. Li, L. D. Xu, and S. Zhao, "The internet of things: a survey," *Information Systems Frontiers*, vol. 17, no. 2, pp. 243–259, 2015.

[4] L. Atzori, A. Iera, and G. Morabito., "The internet of things: A survey," *Comput. Netw., vol. 54, iss. 15, pp. 2787–2805*, 2010.

[5] Gartner, "Typical Family Home Could Contain More Than 500 Smart Devices by 2022," 2020. [Online]. Available: https://tinyurl.com/gartnerSmartDevices

[6] L. Yao, Q. Z. Sheng, and S. Dustdar, "Web-based management of the internet of things," *IEEE Internet Computing*, vol. 19, no. 4, pp. 60–67, 2015.

[7] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE communications surveys & tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[8] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[9] S. Li, L. Da Xu, and S. Zhao, "The internet of things: a survey," *Information Systems Frontiers*, vol. 17, no. 2, pp. 243–259, 2015.

[10] Juniper Research, "IoT connected devices to almost triple to over 38 billion units by 2020," 2019. [Online]. Available: https://tinyurl.com/juniperresearchIoT

[11] C. S. Jensen, H. Lu, and B. Yang, "Indoor-a new data management frontier." *IEEE Data Eng. Bull.*, vol. 33, no. 2, pp. 12–17, 2010.

[12] R. C. Shit, S. Sharma, D. Puthal, and A. Y. Zomaya, "Location of things (lot): A review and taxonomy of sensors localization in iot infrastructure," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 2028–2061, 2018.

[13] D. Zeinalipour-Yazti, C. Laoudias, K. Georgiou, and G. Chatzimilioudis, "Internet-based indoor navigation services," *IEEE Internet Computing*, vol. 21, no. 4, pp. 54–63, 2017.

[14] P. Bolliger, "Redpin-adaptive, zero-configuration indoor localization through user collaboration," in *Proceedings of the first ACM international workshop on Mobile entity localization and tracking in GPS-less environments*, 2008, pp. 55–60.

[15] Find, "Framework for Internal Navigation and Discovery." [Online]. Available: https://www.internalpositioning.com

[16] D. Zeinalipour-Yazti and C. Laoudias, "The anatomy of the anyplace indoor navigation service," *SIGSPATIAL Special*, vol. 9, no. 2, pp. 3–10, 2017.

[17] Y. Gu, A. Lo, and I. Niemegeers, "A survey of indoor positioning systems for wireless personal networks," *IEEE Communications surveys & tutorials*, vol. 11, no. 1, pp. 13–32, 2009.

[18] A. Konstantinidis, P. Irakleous, Z. Georgiou, D. Zeinalipour-Yazti, and P. K. Chrysanthis, "Iot data prefetching in indoor navigation SOAs," *ACM Transactions on Internet Technology (TOIT)*, vol. 19, no. 1, pp. 1–21, 2018.

[19] A. Konstantinidis, G. Chatzimilioudis, D. Zeinalipour-Yazti, P. Mpeis, N. Pelekis, and Y. Theodoridis, "Privacy-preserving indoor localization on smartphones," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 11, pp. 3042–3055, Nov 2015.

[20] C. Rizos, A. G. Dempster, B. Li, and J. Salter, "Indoor positioning techniques based on wireless LAN," in *1st IEEE Intl. Conf. on Wireless Broadband and Ultra Wideband Communications*. IEEE, 2007, pp. 13–16.

[21] Apple, "Ultra Wideband is available on iPhone 11 devices," 2020. [Online]. Available: https://tinyurl.com/appleUWB

[22] LoRa, "Low-power wide-area network technology." [Online]. Available: https://lora-alliance.org

[23] RobotOS, "Powering the world's robots." [Online]. Available: https://www.ros.org

[24] P. Framework, "The High Velocity Web Framework For Java and Scala." [Online]. Available: https://www.playframework.com

[25] D. Lymberopoulos, J. Liu, X. Yang, R. R. Choudhury, V. Handziski, and S. Sen, "A realistic evaluation and comparison of indoor location technologies: Experiences and lessons learned," in *Proceedings of the 14th international conference on information processing in sensor networks*. ACM, 2015, pp. 178–189.

[26] HAProxy, "The Reliable, High Performance TCP/HTTP Load Balancer." [Online]. Available: http://www.haproxy.org

[27] MongoDB, "Cross-platform document-oriented database." [Online]. Available: https://www.mongodb.com

[28] InfluxDB, "Open-source time series database." [Online]. Available: https://www.influxdata.com

[29] CouchbaseDB, "Distributed multi-model NoSQL document-oriented database." [Online]. Available: https://www.couchbase.com