MEDInA

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts
Western Switzerland

MEDInA: **sMart EDge fabric for Iot Applications**

WP4: Validating InaaS architecture with the IoT data integrity use-case (Trust Noise)

Final Report

February 2022

Authors:

Francisco Mendonca, Nabil Abdennadher (HES-SO//Geneva), Paul Royo, Arianna Religi (SABRA)

1. Introduction

MEDInA is an Innosuisse that aims at designing an intelligence as a Service (InaaS) framework for creating IoT self-adaptive Machine Learning (ML) based applications, and deploying them on a shareable backbone infrastructure composed of three layers: IoT objects, edge devices and cloud infrastructure [1].

InaaS is evaluated in a real concrete case of a self-adaptive application called TrustNoise. This work is carried out within Work Package 4 (WP4) of the project. The objective of TrustNoise is to assess the integrity of the data collected from the 900 sensors deployed in the Carouge area. While these sensors do not meet the class 1 requirements (IEC 61672-1), their primary purpose is measuring road traffic noise, with levels ranging from 40 to 80 dB. These sensors demonstrated a measurement difference of less than 0.5dB, in real traffic conditions, when compared with class 1 sonometers. They report data every 15 minutes, at different sound levels (Lmax, Lmin, Leq, L95, L90, L50, L10), through a LoRaWAN network. These noise sensors measure sound pressure levels in dB(A).

To ensure finer acoustic definition and to establish redundancy, in some hot spots, more than one sensor was deployed, at different heights on the same vertical axis. In all areas, sensors were deployed at a 3-meter height. In certain hot spots, they are also deployed at heights of 6 and 9 m (Figure 1).

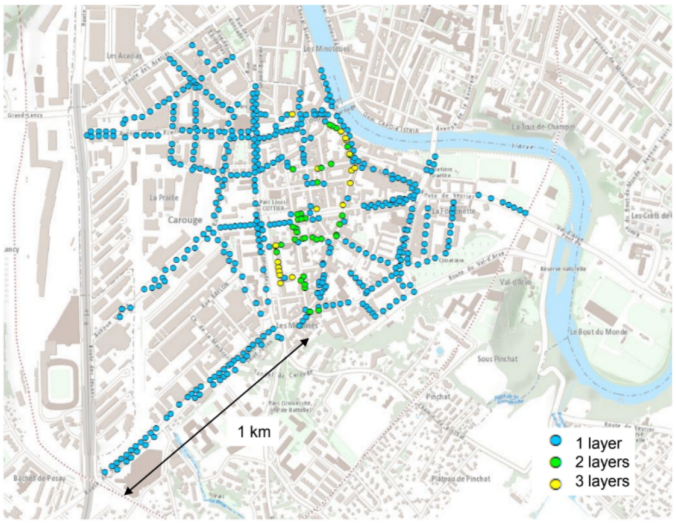

Figure 1: Sensor Deployment

IoT devices are often subject to misbehavior resulting from the poor quality/performance of devices or from external disturbances.

The goal of this Trust application is to detect misbehaving sensors and assign a low trust score to the data they are sending.

This Work Package (WP4) has the objective of evaluating the integrity of the data gathered by sensors. As these sensors are of inferior quality to what the law requires, we devised a Trust Model, to deduce the reliability of a sensor.

2. Trust Model

As some sensors might be at fault or measure in particular conditions, the data acquired by them might be inaccurate or non-representative. For example, when manifestations occur, strong precipitation, construction, and other obstructions might distort the average noise of the area. To solve this issue we will use a Trust Model, to infer the trust of the data.

An Interaction is a set of data points that corresponds to a time frame. In this case, an interaction represents one day, and is composed of a set of 96 data points, as each sensor sends a data point every 15 minutes.

A Trust Model is a middleware that detects when a sensor is misbehaving and tags it accordingly. The foundation of the Trust Model, and how it works has been described in [2]. Generally speaking, to calculate the trust of interactions gathered by a given sensor, we represent sensors by a set of parameters, coined Trust Factor (TF). Each TF expresses a given characteristic of the sensor that could affect its trustworthiness. The nature and number of TFs depend on the sensor's type. For each interaction, a score related to a given TF is processed. A global data quality score, modeling the trustworthiness of the interaction, is then calculated.

In the previous work [2], we pre-selected four Trust Factors: Availability, Accuracy/Precision, Clustering, and Packet Error Rate (PER). Of these four, PER was excluded, as this factor is guaranteed by the network (LoRaWAN).

In the following sections, we will describe the work done on the Accuracy/Precision and Signature Trust Factors. The Availability Trust Factor hasn't changed since the last report.

2.1 Accuracy/Precision Trust Factor

The Accuracy/Precision Trust Factor represents how close interactions are from its neighboring sensors. This serves to deduce if the interaction obtained by a sensor is close to a "perfect" sensor in the same conditions.

Assuming that a majority of the sensors behave correctly, it is to expect that neighboring sensors, being sensors that share similar contexts, should report similar measurements. Therefore, we can create a confidence interval to detect if values reported by a particular sensor are within this interval.

Neighborhood “Model”

We use a simple radius-based approach to calculate the neighbors. In this approach, for each sensor was drawn a circle with a fixed radius (e.g 30 meters), and any sensors that are inside this radius are classified as neighbors.

This approach is easy to calculate, but it doesn’t take into consideration the context of each sensor. A sensor S1, that is placed in an avenue, has a mostly different noise signal than sensor S2 placed in a small pedestrian street and close de S1. An alternative to the radius-based approach is to use an intersection between the radius-based neighboring model, and the output from the signatures created for the Signature Trust Factor (see detail below), thus taking into consideration the context of the sensors.

To calculate how close S1 is to its neighbors, first, a set of close sensors is made, using the neighborhood model. Then, the sensor interaction is compared with the interactions of its neighbors, using a simple distance metric Finally, an average of all the comparisons is made The closer the interactions, the higher the output will be. Finally, an average from the comparison between the sensor and its neighbors is retrieved.

2.2 Signature Trust Factor

The Signature Trust Factor represents the context a sensor belongs to. It is composed of environmental, positional, and acoustic conditions of the sensor location. Sensors belonging to the same context should have similar signatures.

Since the streets, where the sensors are deployed, have different configurations (avenues or small streets; cars or no cars, 30 km/h zones; etc.), and the sensors are installed in different heights, the signature of a sensor might be different than the surrounding sensors. By clustering the sensors by their signatures, we can determine if the interaction is similar to the expected signature.

Signature generation must rely on “representative” data recorded by noise sensors. To create the signatures, first, the data collected by the sensors is filtered, and then the filtered data is clustered. Each cluster represents a given signature.

Filtering

The objective of the filtering procedure is to create a clean dataset, to then build a baseline used to cluster. The baseline should only be composed of “representative” data.

To do so, we need to discard noisy data. Our approach is straightforward: when an anomaly is encountered, the full day is marked as “non-representative”. The procedure used to filter has been described in [1].

Clustering

A clustering algorithm tries to classify sensors into specific “contexts” (called signature). For this purpose, we use unsupervised techniques. Based on meaningful data, the clustering algorithm aggregates, in clusters, the most similar sensors. The algorithm used is K-Means [3], as it is simple to implement, and is efficient to use large datasets with it.

A Signature can be described as any component that benefits the characterization of the sound. So, the Signature used is defined by the average interaction. As each day of the week has its pattern, we produced a set of signatures for each sensor and for each day of the week.

To start the clustering, first, we do an “Elbow” Analysis (Figure 1), which consists of finding the correct number of clusters, for the given dataset.

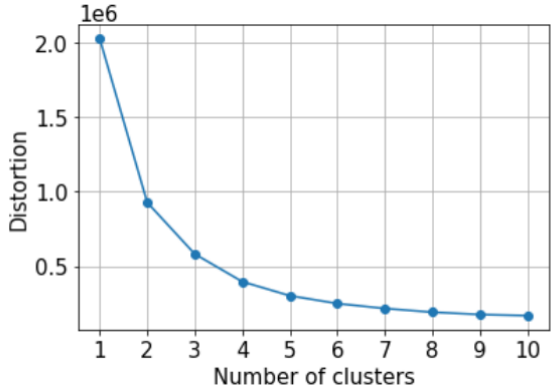

Figure 2: Elbow Analysis Graph

The Elbow Analysis indicates the best number of clusters, for the given data, by discovering the inflection point in the curve. This is done by clustering with different numbers of clusters and then analyzing the distortion of the model. Using the results from Figure 1, the best number of clusters is 3 or 4. While 3 or 4 clusters might be the best number of clusters, it might not be enough to describe the several different contexts that a city might have. As such, we used 5 clusters.

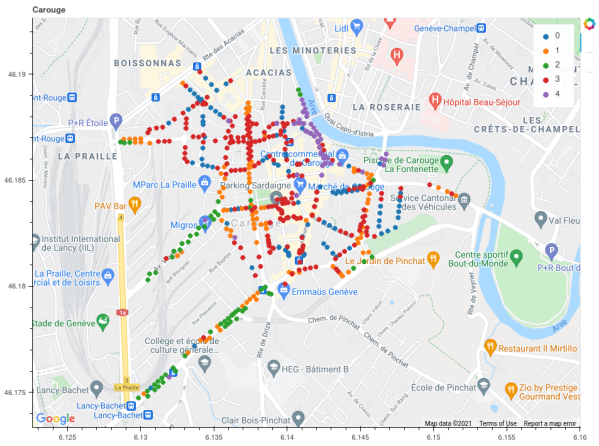

Figure 3: Clustering Map for a Single Day (Monday)

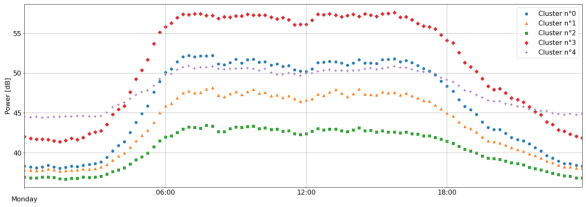

Figure 4: Cluster Distribution for a Single Day (Monday)

The output of the clustering should provide a map where most sensors are correctly grouped by area - avenues, small streets, near the river, near the train station, as can be seen in Figure 3. Each color on the map corresponds to a different cluster and, each dot to a different sensor. In Figure 4 can be seen the differences in the clusters, by decibels.

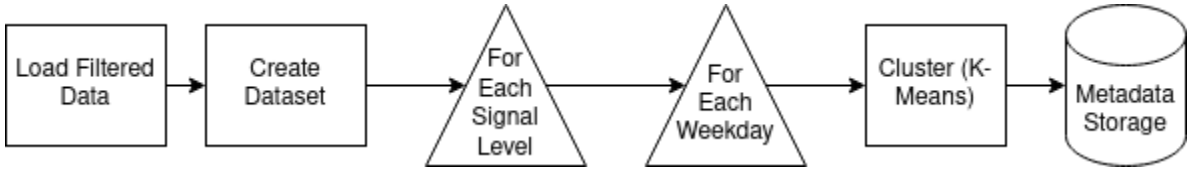

Figure 5: Clustering Workflow

Figure 5 describes the workflow used to cluster. In this Figure, the rectangular-shaped objects are functions, the triangular-shaped objects are loops done on the data, and the cylinder-shaped object is where the Clustering objects are stored, so that later when running the Trust Factor, these Clustering Objects can be retrieved and used.

For this use case, each sensor has seven signatures, one for each day of the week. This is because each day has different sound signatures, and this difference is even more evident on the weekend.

Trust Factor

When calculating the Signature Trust Factor, the clusters produced previously are used to evaluate the incoming interaction. To start, a prediction is made to identify to which cluster the interaction belongs. If this cluster is different from the Sensor Cluster, then the Signature Trust Factor score is 0, as the

incoming interaction is severely different from the Signature from the sensor. If the interaction belongs to the same cluster, and thus, it's similar to the Sensors Signature, then a simple Euclidean Distance is used to calculate the distance between the interaction and the Cluster Centroid. Finally, the difference between the average distance, and 1 is used as the Signature Trust Score.

3. Deployment

3.1 Trust Model

As specified earlier, the trust model relies on daily data and each day of the week has its own signal. As a result, trust scores updates must be carried out daily. As the Trust Model uses clusters and handles a fair amount of data, a VM on a cloud provider would be ideal. The process, even for powerful machines, is fairly intensive, so a cronjob would start the Trust Model every day. The output of the Trust Model needs to be stored in a separate database from the interactions. In another database or object store, the cluster files and the neighborhood models need to be stored.

To start, a Clustering process (including the Filtering) needs to be run, and a Neighborhood model needs to be created. After analyzing the data generated by the Trust Model, if any misbehaving sensors are found, a decision should be made whether the sensors in question are misbehaving, or if the clusters aren't ideal anymore. Either way, after taking the proper precautions, the clustering and the neighborhood model processes should be re-run. For this, the proposed architecture is presented in Figure 6.

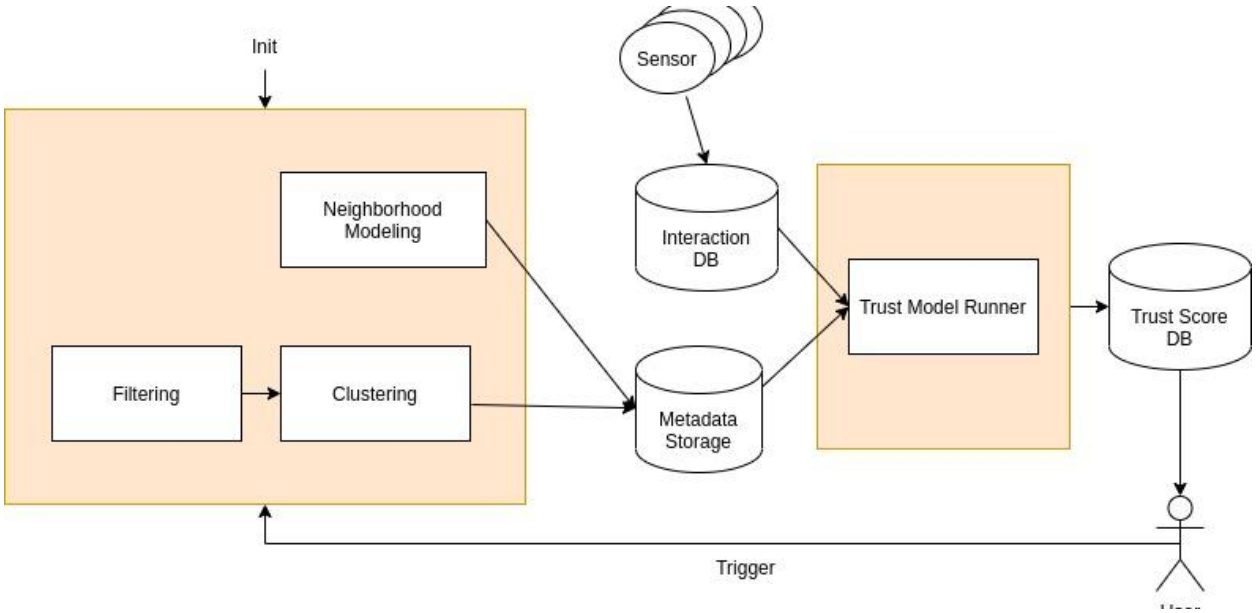

Figure 6: Full System Architecture

The Metadata Storage stores the weights for the clustering and the neighborhood files, used in the Signature and Precision Trust Factors, respectively.

Also part of the Trust Model is the visualization module, to analyze the results. The visualization module is in the same interface as the filtering interface and accesses directly the database where the results from the Trust Model are stored.

The Filtering Process, as used in the Web Interface, is composed of several functions that, when called, execute different steps. The previous implementation of the Filtering Process was using a web server that was kept in a considerably large Cloud instance. This instance was large to utilize parallel execution of functions, to speed up the process of filtering. As the filtering process isn't used every day, and the VM, to be accessible at any given time, needs to be up all the time, one solution we thought of was the use of Function as a Service (FaaS) paradigm. When using FaaS, the cost of using the service is only actioned when the function is called, and so the service is only paid when in use.

3.2 Lambda Functions

AWS Lambda is a Function as a Service (FaaS) service, which allows running pieces of code without provisioning or managing servers. It's event-driven, meaning when an event is triggered (HTTP request, MQTT message, API call, or other) a function associated with that event is run. The code can be either packaged as a ZIP file or as a Docker container and, in the latter case, is stored on AWS Container Registry. AWS Lambda is supposed to be used with small functions that, usually, trigger other events, and not for large workloads.

As we noted previously, we used a large instance to utilize parallelization when filtering. AWS Lambda isn't, by default, able to parallelize work in a single function. To overcome this problem, the functions that need to run parallel are called several times, with different entry data. Then, the results of the parallel functions are gathered in the main function and returned to the user.

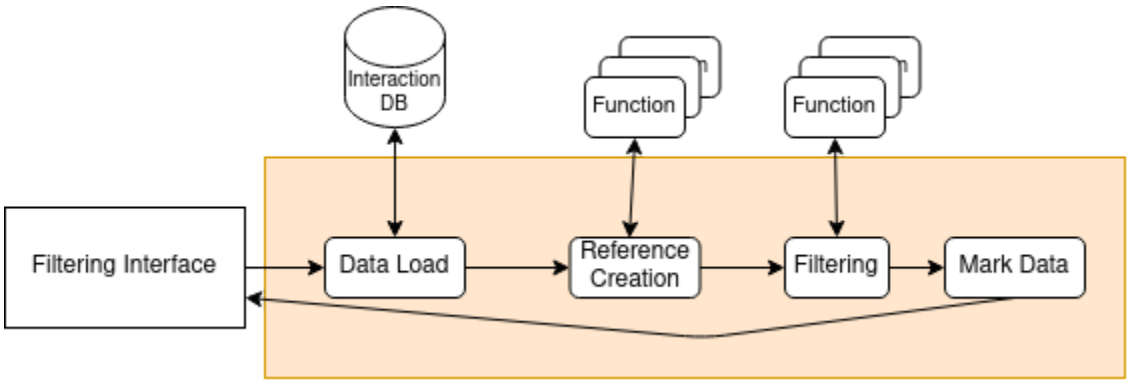

Figure 7: AWS Lambda Architecture for Filtering

With this architecture (Figure 7), when the filtering interface makes an HTTP request, it will be consumed and processed in the Entry Point, which is managed by AWS. Then, trigger a Lambda function (Data Load) that will read the interactions from a DB, and launch a Lambda per weekday, to create a reference. Then, once the references are built, it will trigger as many Lambdas as days that will be filtered. To this, the correspondent reference is passed as the input data. Finally, a final lambda is triggered to mark the data and return the data to the interface.

The result of the Filtering Functions can be analyzed in the Filtering Interface. Figure 8 represents the starting page of the Filtering Interface, where the user selects a sensor. Figure 9 is a calendar created from the filtered data, and Figure 10 is the representation of the resulting data from the filtering, for a selected day.

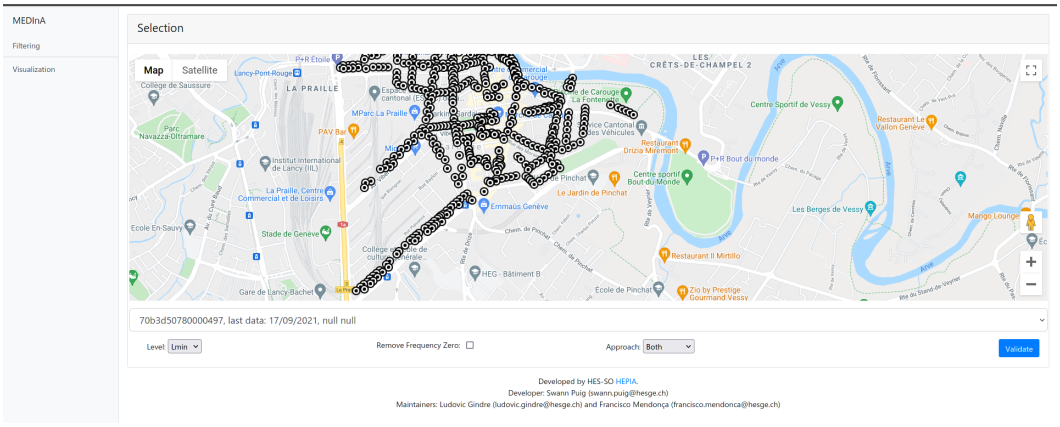

Figure 8: Sensor Selection

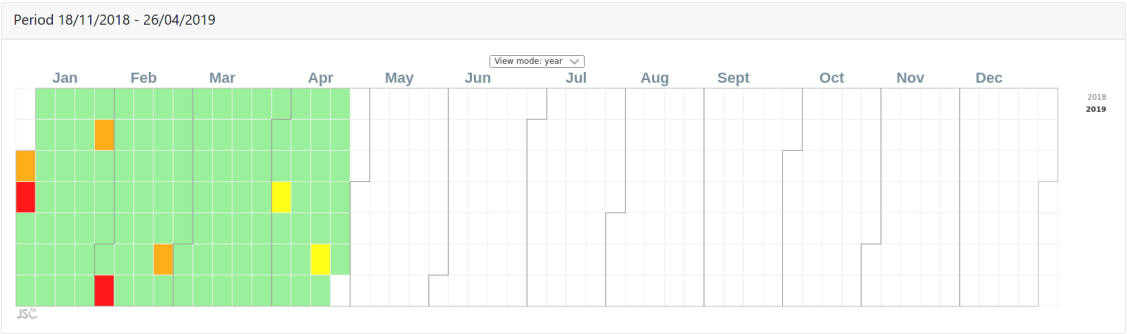

Figure 9: Date Selection

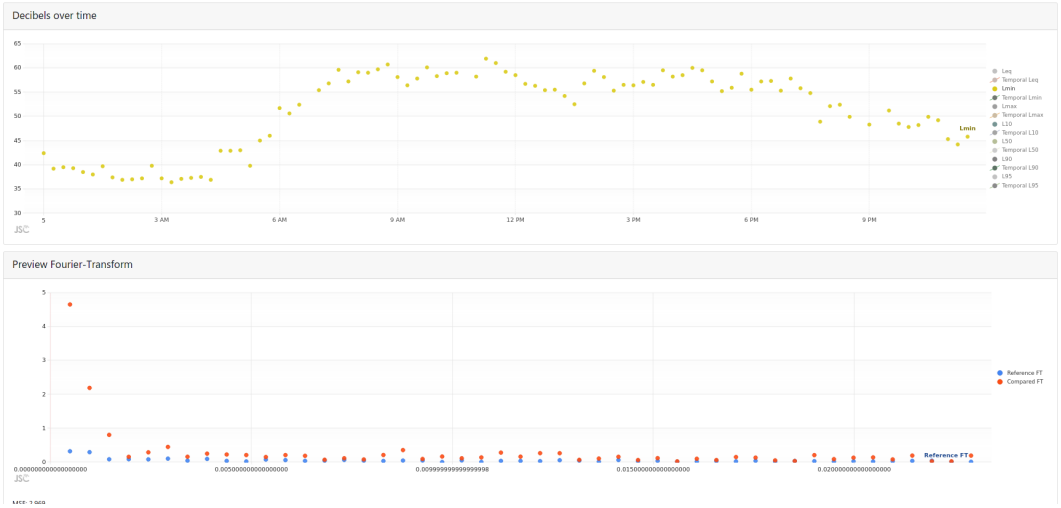

Figure 10: Frequency and Temporal Analysis

4. Conclusion

In this project, we were faced with an increasingly common issue within the IoT space: How to deal with unreliable devices. This is especially common for non-critical applications, where the data that these devices produce can only be observed. As such, there is a need to assess if the data that these devices are reporting is trustworthy.

To that end, in this work package, we introduced a Trust Model that uses several metrics, employing both spatial and temporal correlations in the data produced by the sensors that were deployed in the Carouge area.

In this Trust Model we use a Signature-based approach, where sensors are clustered by their signature. A signature is a definition of the sensor's context - where the sensor is placed, the weather, the season, and others. Using the signature, we are able to filter data, in order to prove that the data coming from the sensor is valid.

To test the signatures, two different configurations were tried: the first was done using the average reports for a week period, and the second the average reports for one day. For the second, a set of signatures was created for each day of the week. The tests consisted in evaluating the trust model in the case of the two configurations presented above. The approach is as follows: we have reserved 30% of the collected and labeled data (good or bad) to evaluate the performance of the trust model. In the case of the first configuration, the results are disappointing: the score generated by the model is consistent with reality in only 2% of cases. The second configuration provided better results: between 33 - 43% accuracy. These results are better than those of the first configuration. However, they are still far from an acceptable success rate. As an improvement, we are currently working in two directions: the first is to

identify time periods that better reflect the behavior of the noise IoT sensors. The second is to consider other clustering algorithms than k-means. Indeed, these mitigated results are probably due to the variability and the size of the data. In the future, we will consider tools such as XGBoost [4] and Long Short Term Memory (LSTM) [5]. Although not a clustering method, XGBoost is a classifying method that could be used to identify the class of a particular sensor.

The experimental results of the signature-based trust metrics still need to be improved. On one hand, our day-of-week signatures are possibly too generic for robust trust estimations in such chaotic scenarios as those of busy urban areas. On the other hand, the sparsity of our dataset prevents refining our signatures for a better fit: ideally, each day of the year might be specific enough to warrant its own signature, however, that would require several years' worth of data. As for the signature extraction: large clusters carry both the benefit of a leaner trust computation phase and the risk of underfitting. Thus, it looks reasonable to do further experiments with more (w.r.t. their optimal number) smaller clusters, or even with no clustering at all, namely with sensor-specific signatures.

Finally, The MEDInA project has developed a Trust framework to test new Trust Factors, refine the Trust Factors already identified, filter data where appropriate and set up the aggregation function used to calculate the global Trust Score.

As part of this work package, we also showed how this Trust Model can be deployed in several types of architectures - Cloud-VM or Lambda Functions, so that it can fit the requirements of the end-user.

5. Bibliography

1. Visuelle C-C. sMart EDge fabric for Iot Applications. [cited 6 Feb 2022]. Available: https://lsds.hesge.ch/smart-edge-fabric-for-iot-applications/

2. [No title]. [cited 11 Oct 2021]. Available: https://lsds.hesge.ch/wp-content/uploads/2021/06/MEDInA_Midterm_Report.pdf

3. 2.3. Clustering. [cited 12 Oct 2021]. Available: https://scikit-learn/stable/modules/clustering.html

4. XGBoost Documentation — xgboost 1.5.2 documentation. [cited 2 Feb 2022]. Available: https://xgboost.readthedocs.io/en/stable/

5. Long Short-Term Memory. [cited 2 Feb 2022]. Available: https://ieeexplore.ieee.org/abstract/document/6795963