We integrated into the platform some anomaly and intrusion detection techniques exploiting the network statistics gathered from the infrastructure. Our system is composed by four parts: rule-based, statistical-based, machine learning techniques, and streaming machine learning approached.

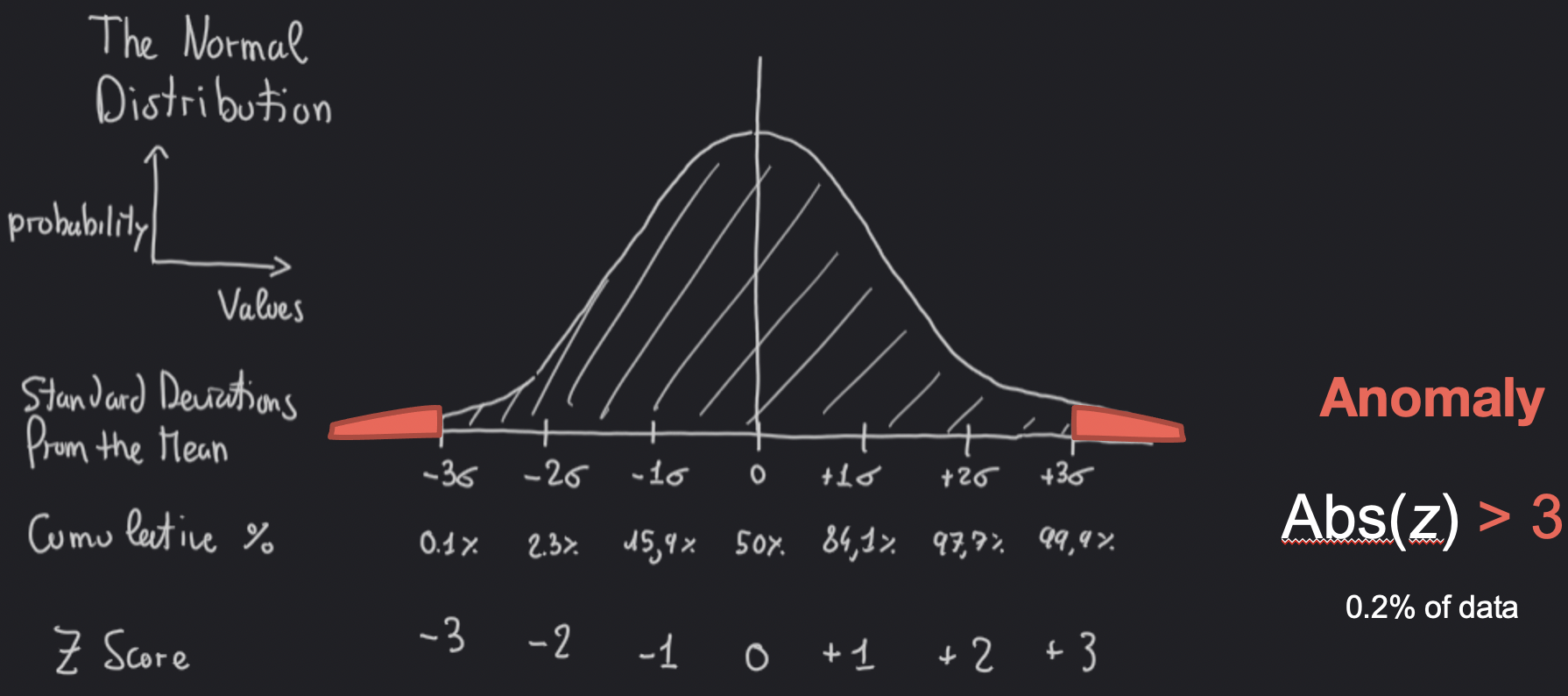
# Rule-based

This is a high-grained task, aiming at reporting any new network transactions and raising an alert. With a certain frequency, we check if there are any new source and destination addresses, source and destination ports, services, and API called with respect to the check before. If so, we create an alert into the platform.

# Statistical-based

This is a more fine-grained task, aiming at using the number of bytes and packets created, and the total bytes and packets sent (measurements) throw the network to identify anomalous transactions. A common technique to find anomalies is to calculate the Z-score. It is a statistical measure that tells how far is a data point from the rest of the dataset, supposing that it follows a normal distribution. In a more technical term, Z-score tells how many standard deviations away a given observation is from the mean. Its formula is:

If the Z-score of a transaction is higher than a threshold, it means that it is an anomalous transaction. For example, if we use a threshold equal to 3, it means that we are considering as anomalies the 0.2% of the data (0.1% in the left tail and 0.1% in the right tail).



For each measurement, we group the data by source and destination address, source and destination port, service, protocol, and API called creating a sort of unique identifier called *hash.* For all the transactions having equal *hash* we calculated and stored their mean and the standard deviation values.   
Then, with a certain frequency, we retrieve all the network transactions, and, for each of them, we calculate the Z-score using the corresponding mean and the standard deviation values. If the Z-score is greater than the selected threshold, that transaction is considered anomalous. Finally, we incrementally update the mean and standard deviations values using only the new non-anomalous transactions. To do that, we use the following formulas:

# Machine Learning Techniques

To have a more fine-grained result, with a localized number of anomalies, we adopted some machine learning techniques. The main challenge in network anomaly detection for the Internet of Things was the lack of labelled data; due to this, we used unsupervised anomaly detection techniques – by assigning a score to each data element proportional to its abnormality with respect to the rest of the data set. Such techniques do not require a training phase per se, in the sense that they do not try to optimize a function using the use cases provided in the training set, in fact - each analyzed data element becomes part of the model used to evaluate other elements, typically require multiple passes over the data set to evaluate triggers.   
The first method used is the Local Outlier Factor (LOF). It is a density-based algorithm that requires the computation of pairwise distances in the data set. However, contrary to distance-based methods, the anomaly score of LOF depends on the difference between the neighborhood density of a data element and that of its k nearest neighbors.

The second method tested is the Isolation Forest (IForest), an ensemble anomaly detection method. Ensemble methods draw from the idea that an ensemble of models, learned from variations of the same data set, can perform better than a single model learned from the data set as a whole. The anomaly score from the ensemble is obtained by aggregating the scores of its component using an aggregation function, such as the mean or maximum. In particular, IForest consists of a forest of Isolation Trees. The technique detects anomaly by isolating each element in a data set with axis-parallel random cuts. The anomaly score is inversely proportional to the number of cuts necessary to isolate an element, implementing the idea that elements in low density regions can be isolated with a lower number of cuts.

The last method tested is the One Class Support Vector Machine (OneClassSVM). SVMs use hyperplanes in multi-dimensional space to separate one class of observations from another. It is called OneClass because, in this anomaly detection case, we do not have different classes, but only normal or anomalous transaction.

However, since we are dealing with non-stationarity data that can change over time, we should continuously update the models from scratch every time new transactions are available. This causes a waste of time, memory, and possibly misclassified transaction.

# Streaming Machine Learning Techniques

This is the reason why, for the last step, we used some Streaming Machine Learning techniques. They tackle time, memory, and non-stationarity problems that affect traditional ML methods. Ideally, every time a new instance arrives, a streaming learner inspects it but without saving it in memory. Then the model is updated incrementally, being able to predict at each moment. In this way, the algorithm avoids data storage problems because it discards the new instance immediately after the training phase. The time problem is addressed by updating the model incrementally, one instance at a time, without the need to retrain it from the beginning. Additionally, the very same approaches can detect when non-stationarity occurs and adapt the model accordingly.

We tested the streaming version of both the OneClassSVM and IForest methods. With a certain frequency, we retrieve all the network transactions, and we update the models with each of them, also detecting if it is an anomalous transaction or not.

# Infrastructure

Spiegare perchè abbiamo usato Flask server

Architettura back-end (Riot connector e Influx connector)

Come ci colleghiamo al server di RIoT

# User guide

To use these services, we incapsulated them into a docker container and we exposed some Rest API to visualize and analyze the anomalies achieved.

### Dockerization

### Rest API

To visualize the results achieved, we create a back-end server with Flask that exposes the following Rest APIs:

*GET/status*

It returns the log with the last errors occurred

*GET /<measurement>/stats*

For the measurement selected (tdmp\_bytes\_created, tdmp\_bytes\_total, tdmp\_packets\_created, tdmp\_packets\_total), it returns the hash list with all the fields and values of which it is composed.

*GET /<measurement>/stats/<hash>*

For the measurement and hash selected, it returns the current value of the statistics used to calculate the Z-score value (mean, standard deviation, squared sum of values, and number of transaction.

*GET /<entity\_type>*

For the entity\_type selected (dst, dstp, src, srcp, service, and url), it returns the list with all the values seen.

*GET /<measurement>/anomalies*

For the measurement selected, it returns all the anomalies detected through the statistical-based approach.

*POST /<measurement>/anomalies/<uuid>*

Through this POST is possible to confirm if a specific transaction (uuid) is really an anomaly or not. If It is not, it will be considered as normal transaction and so removed from the anomalies list. Moreover, it will be used to incrementally update the statistics used to compute the Z-score.