# Data description

Data can be accessed from an InfluxDB server, which allows an initial granular and tabular display of the information obtained.

Immagine che contiene testo, monitor, elettronico, nero

Descrizione generata automaticamente

Notably, 4 measurements are available:

* *tdmp\_bytes\_created*
* *tdmp\_bytes\_total*
* *tdmp\_packets\_created*
* *tdmp\_packets\_total*

For each measurement, the variables on which the anomaly detection phase focused are:

* *time*: the timestamp of each traffic connection entry
* *value*: the value of the measurement selected, e.g., for measurement *tdmp\_bytes\_created*, it represents the bytes created for that entry
* *dst*: the IP of the destination
* *dstp*: the port of the destination
* *proto*: the protocol of the connection
* *service*: the service of the connection
* *src*: the IP of the source
* *srcp*: the port of the source
* *url*: the url of the connection

# Anomaly detector

## Methodologies overview

### Rule-based

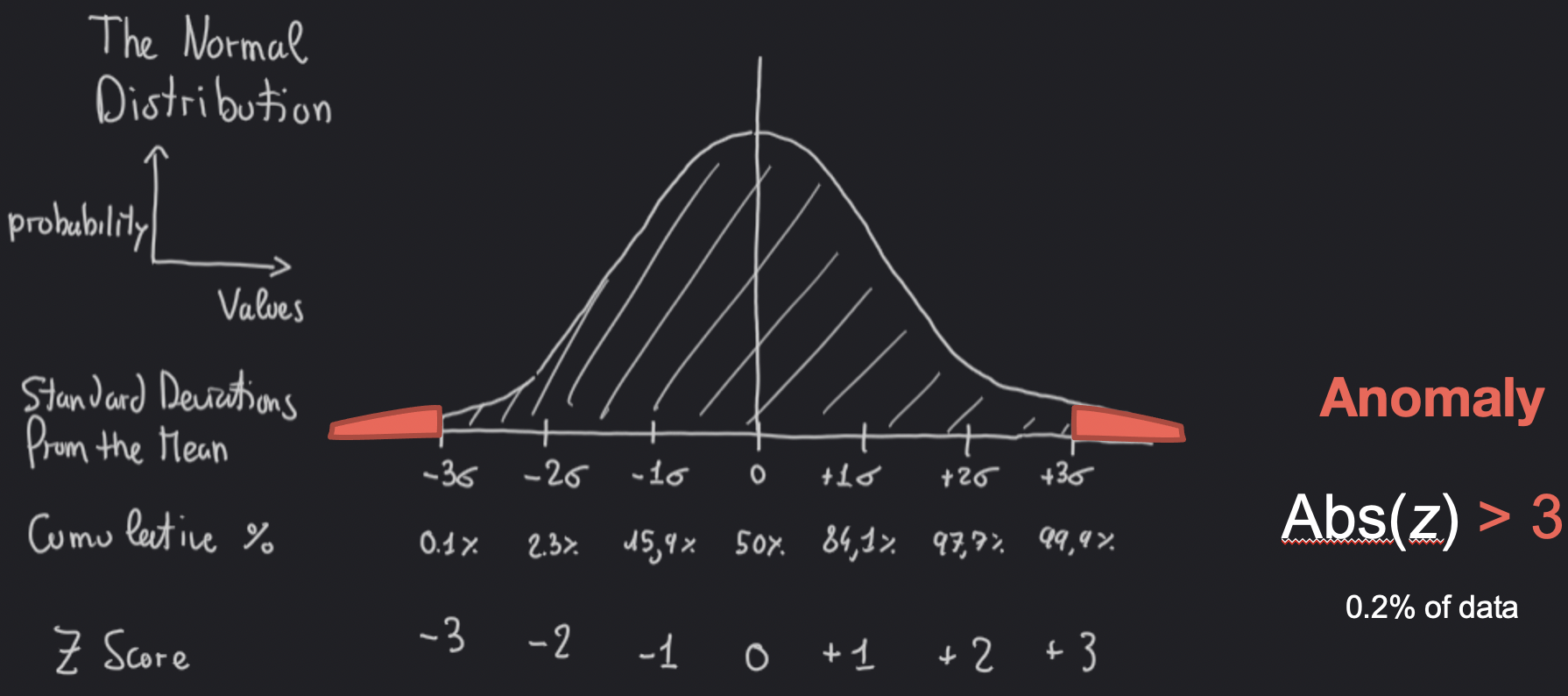
Rule-based packet classification is a powerful method for identifying traffic anomalies in the early phase. Detecting network attack traffic, and non-attack traffic that violates network policy, are two key applications. Many types of unwanted traffic can be identified by rules that match known signatures.

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.

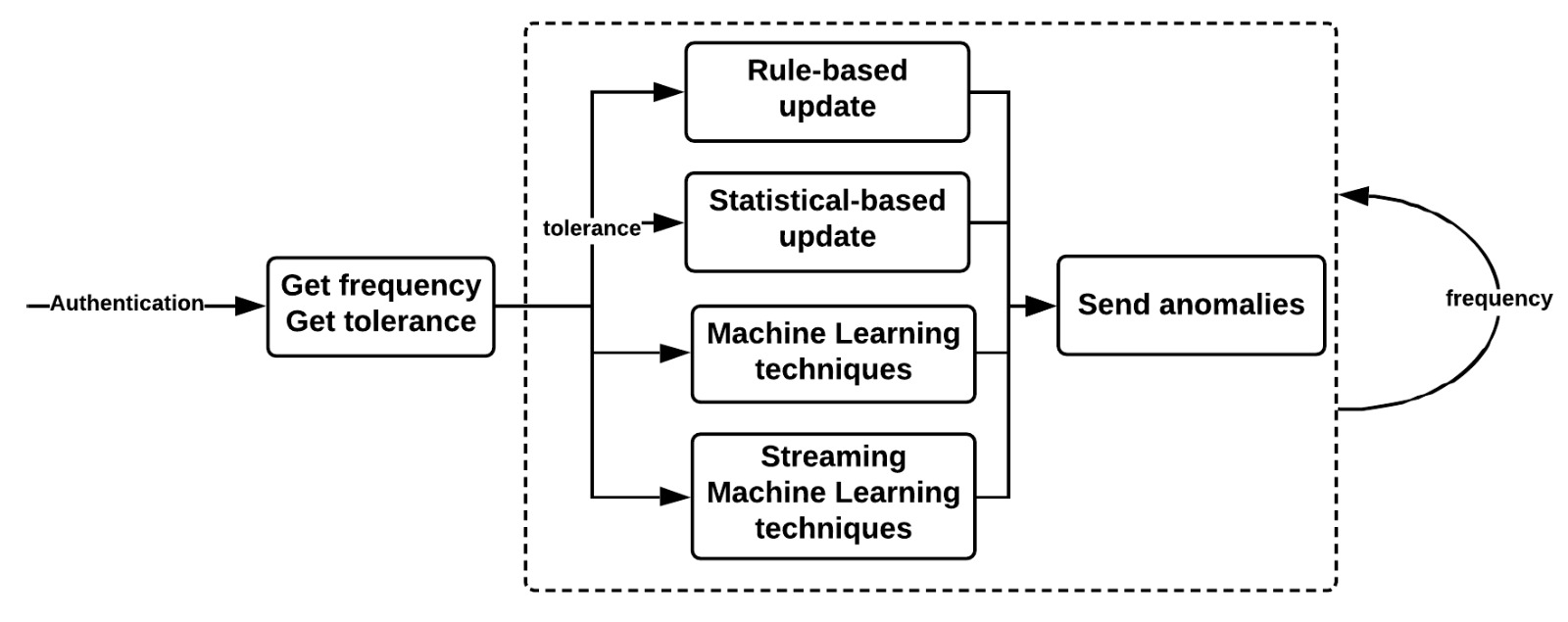
### Streaming Machine Learning Techniques

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.

## Architecture overview

The system we designed for anomaly detection is built around the 4 main anomaly detection techniques described in the previous section, i.e., rule-based, statistical-based, machine learning techniques, and streaming machine learning approaches, which exploit the network statistics gathered from the data.



The anomaly detector needs 2 parameters:

* *frequency*: the frequency (in hours) of the anomalies checking. Example: 24 means that the anomaly detector will search for anomalies every 24 hours on the new data that have been generated from the last check. The default value is 24, the range is 1 – 168 (1 week).
* *tolerance*: the tolerance used by the statistical-based update that regulates the threshold after which an entry must be considered an anomaly, as described in the previous section. The default value is 3, the range is 2 – 6.

Both parameters can be dynamically changed directly on the RIoT server as the anomaly detector system is already configured to get the updated parameters, after the proper authentication.

The four anomaly detection blocks are intended to work in parallel. At each iteration, determined by the frequency, the anomaly detectors get the new data, performs the analysis, and then classify each new entry as an anomaly or not anomaly.

## Back-end Architecture

### Structure

The server is built using Flask. Flask is a lightweight web application framework designed to get results fast and leave room to make the app more detailed in the further developments.

The system relies on two blocks *connectors*. The first one, named *RIoT connector,* connects to the RIoT server to get the parameters and to notify the anomalies found. The second one, named *InfluxConnector*, connects to InfluxDB to get the new data.

#### RIoT Connector

At each function call, RIoT connector creates an authentication, as shown in the figure below, to safely connect to RIoT server.

Immagine che contiene testo

Descrizione generata automaticamente

The authentication is then inserted in the field “Authorization” in the Header of the request. The figure below shows an example of a GET request to retrieve the updated parameters tolerance and frequency, with specific url and base uri (configured at the launch of the application).

Immagine che contiene testo

Descrizione generata automaticamente

#### Influx Connector

Immagine che contiene testo

Descrizione generata automaticamente

The figure below shows an example of use of the Influx Connector. Once the query is ready, the influx connector return the object with all the parameters to do the connection to the InfluxDB.

Immagine che contiene testo

Descrizione generata automaticamente

### Dockerization

### RESTful API

To visualize the results achieved and to monitor the status of the anomaly detector, we create a back-end server with Flask that exposes the following Rest APIs:

*GET /*

It returns a txt file with the possible APIs to see the results.

*GET /status*

It returns the log with the last errors occurred.

*GET /<measurement>/stats*

For the specific measurement selected (i.e., ‘tdmp\_bytes\_created’, ‘tdmp\_bytes\_total’, ‘tdmp\_packets\_created’, ‘tdmp\_packets\_total’), it returns an html table with all the fields and values of which it is composed.

Example: */tdmp\_bytes\_created/stats* returns the html tables of the main statistics of the entries relative to ‘tdmp\_bytes\_created’

*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. Hash represents the unique identifier of a specific entry, which can be retrieved from the table received with */<measurement>/stats*

*GET /<entity\_type>*

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

*GET /<measurement>/anomalies*

For the measurement selected (i.e., ‘tdmp\_bytes\_created’, ‘tdmp\_bytes\_total’, ‘tdmp\_packets\_created’, ‘tdmp\_packets\_total’), 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.

# User guide

To launch the Anomaly Detector, one must execute from terminal the main.py file.

Notably, one must pass to the script **7 command-line arguments, i.e., uri, username, base\_uri, bucket, org, token, url)**. The first 3 represent configuration parameters for the RIoT Connector, while the last 4 refer to parameters for configuring the connection to InfluxDB by the Influx Connector.

Example of executing main.py filer in the anomaly\_detector folder:

python anomaly\_detector/main.py /global service <https://demo.riotsecure.io:6443> riot polimi d2VsY29tZQ== <http://35.152.63.133:8086>

In this case:

* *uri* : /global
* *username* : service
* *base\_uri* : https://demo.riotsecuro.io:6443
* *bucket* : riot
* *org* : polimi
* *token* : d2VsY29tZQ==
* *url* : http://35.152.63.133:8086