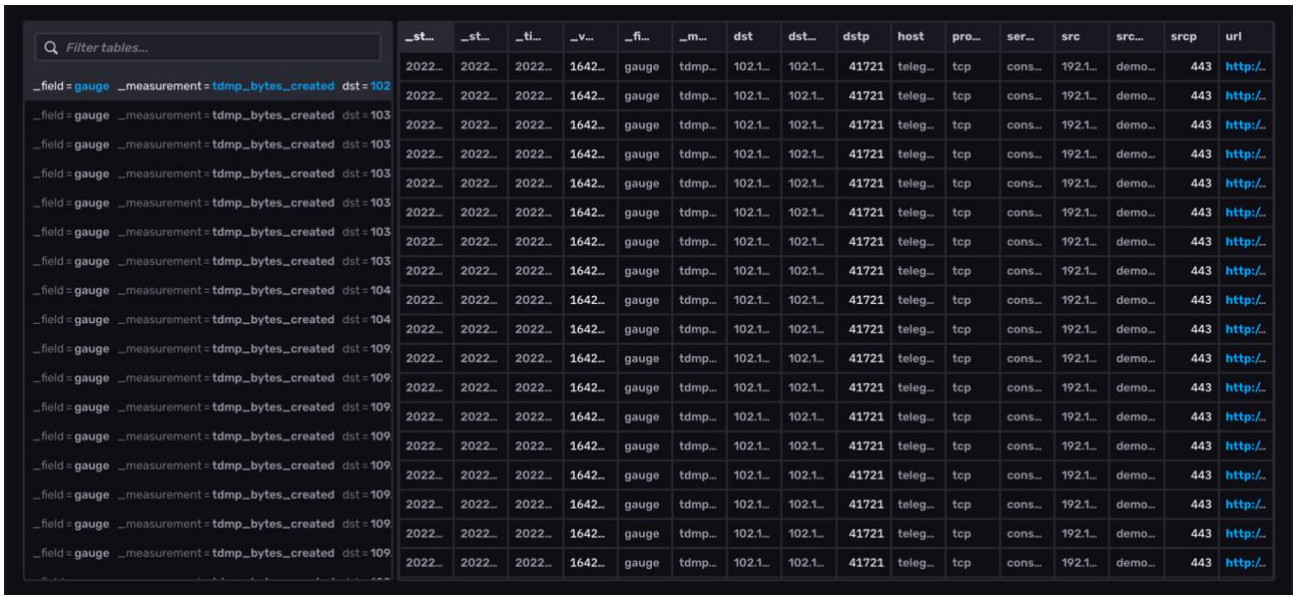
Data description

Politecnico di Milano integrated the anomaly detection methods with the solution developed by RIoT using actual data from the field. In particular, Politecnico di Milano decided to use Telegraph to periodically obtain network monitoring metrics from the Prometheus deployed by RIoT. Furthermore, it opted for InfluxDB as time-series DB to store the metrics and allowing, as the figure shows, an initial granular and tabular display of the information obtained.



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 on 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

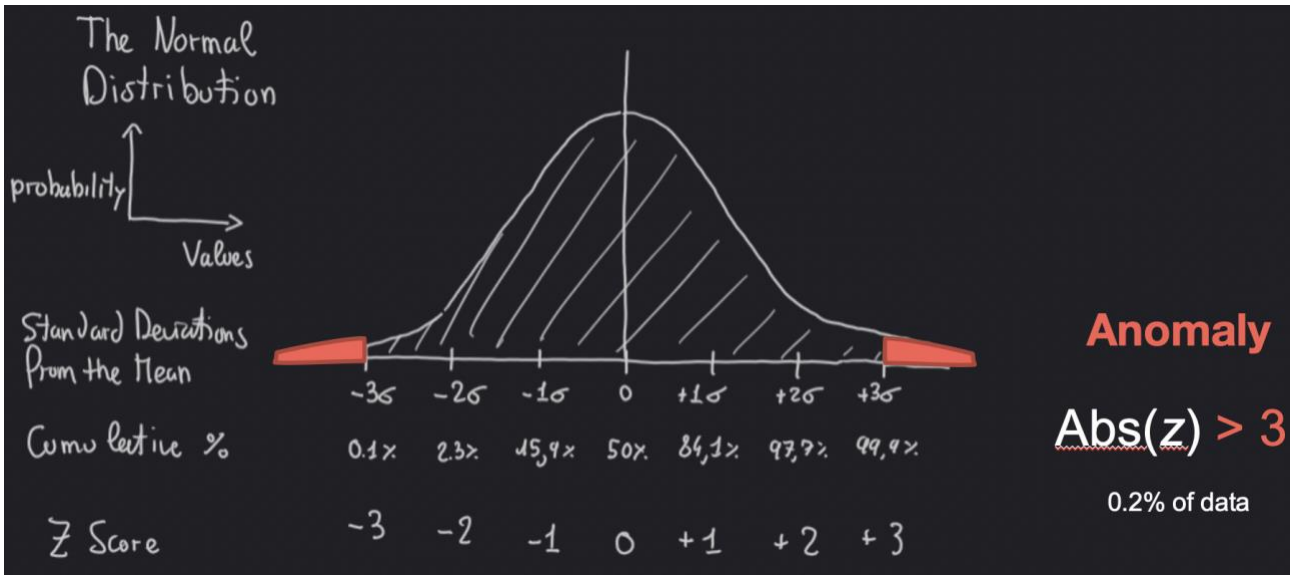
Rule-based packet classification is a powerful method for identifying traffic anomalies in the early phase. Detecting network attack traffic, and non-attack traffic that violates network policy, are two key applications. Many types of unwanted traffic can be identified by rules that match known signatures.
This is a high-grained task, aiming at reporting any new network transactions and raising an alert. With a certain frequency, Politecnico di Milano checks 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, Politecnico di Milano creates 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:

$$Zscore = \frac{\mathrm{x} - \mathrm{mean}}{\mathrm{std}}$$

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, Politecnico di Milano groups 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* Politecnico di Milano calculated and stored their mean and the standard deviation values.
Then, with a certain frequency, Politecnico di Milano retrieves all the network transactions, and, for each of them, it calculates 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, Politecnico di Milano incrementally updates the mean and standard deviations values using only the new non-anomalous transactions. To do that, the following formulas are used:

$$s_i = s_{i-1} + x_i \qquad q_i = q_{i-1} + {x_i}^2$$

$$\mu_i = \frac{s_i}{i} \qquad \delta_i = \sqrt[2]{\frac{1}{i-1} * (q_i - \frac{{s_i}^2}{i})}$$

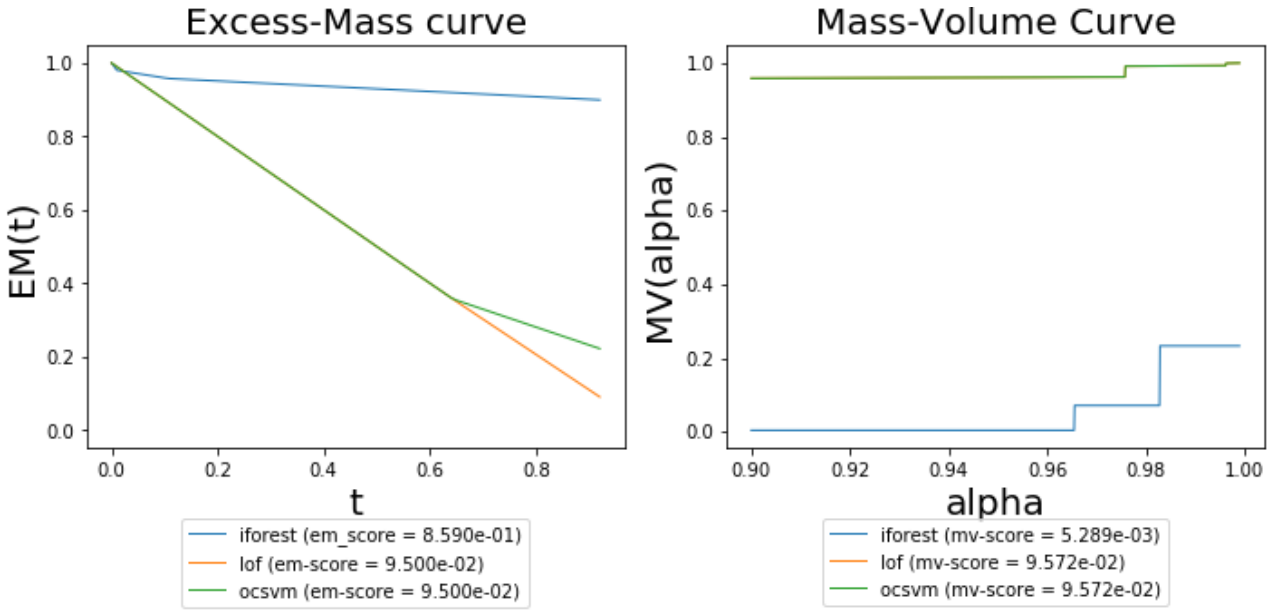
Machine Learning Techniques

To have a more fine-grained result, with a localized number of anomalies, Politecnico di Milano 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, Politecnico di Milano 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 (OCSVM). 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.

Experiments

Politecnico di Milano tested the three methods, i.e., LOF, IForest, and OCSVM, with the data divided for measurements and services. When sufficient labeled data are available, classical criteria based on ROC or PR curves can be used to compare the performance of unsupervised anomaly detection algorithms. Since we did not know which are the true anomalies, we are in unsupervised scenario with no data label. This calls for alternative criteria one can compute on non-labeled data.

Politecnico di Milano opts for using two criteria that do not require labels and shown to discriminate accurately (w.r.t. ROC or PR based criteria) between algorithms. These criteria are based on existing Excess-Mass (EM) and Mass-Volume (MV) curves, which generally cannot be well estimated in large dimension. In particular, a high value of EM and a low value of MV state that a method is better than the others.
The figure below shows the results for the "tdmp_bytes_created" measurement and "postgresql" service. We can notice that IForest method results the best approach.



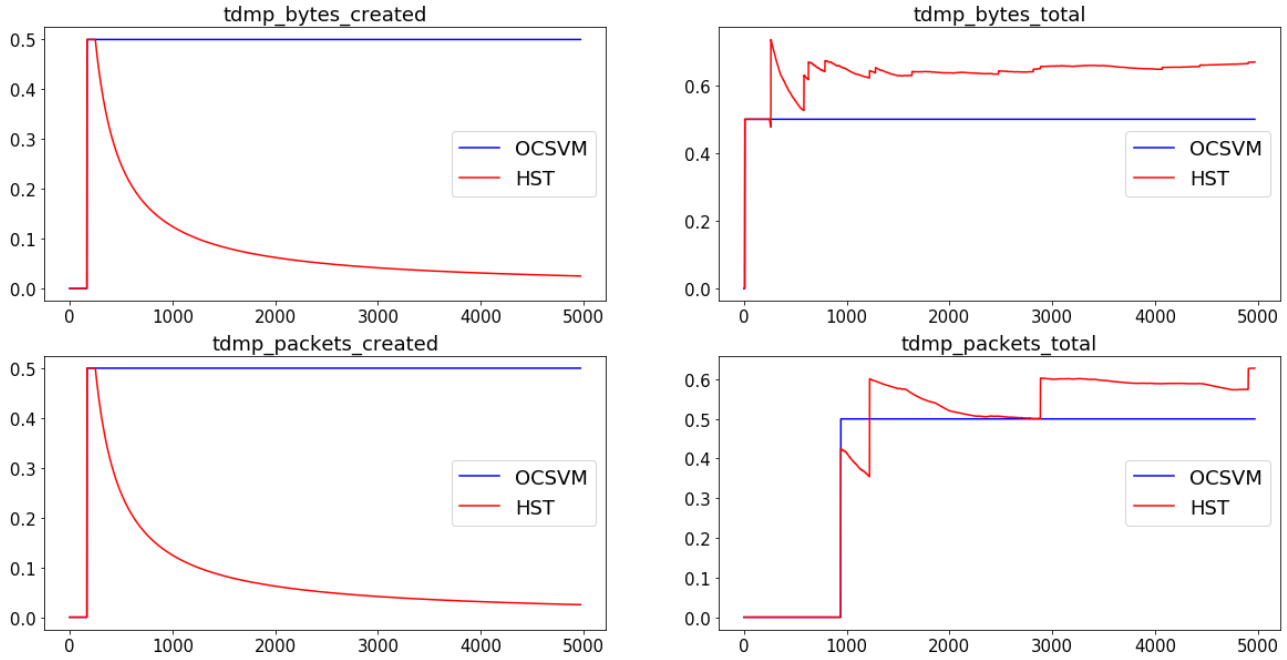
Streaming Machine Learning Techniques

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. This is the reason why, for the last step, Politecnico di Milano investigated 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.
Although the advantages are obvious, there are still clear implementation limitations, as most existing approaches are still batch-oriented. In particular, there are no online metrics for unsupervised anomaly detection that evaluate models in a similar way to the previous section. To overcome this problem, in the experimental phase, Politecnico di Milano used the output generated by the statistical classification techniques as a data label, for the training and evaluation phase of the various Streaming Machine Learning techniques.

Experiments

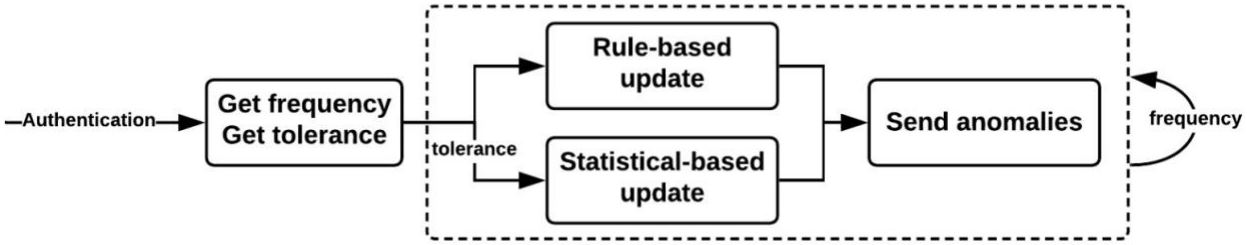
Politecnico di Milano tested the streaming version of both the OCSVM and IForest, called Half-space trees (HST) in SML, methods. The figure below shows the ROC values overtime of the two algorithms tested on the four measurements. In general, no method performs better than the other in

all the experiments. So, Politecnico di Milano needs more tests before deploying them into a production environment.



Architecture overview

The system Politecnico di Milano designed for anomaly detection is built around the 2 main anomaly detection approaches described in the previous section, i.e., rule-based and statistical-based ones, which exploit the network statistics gathered from the data.



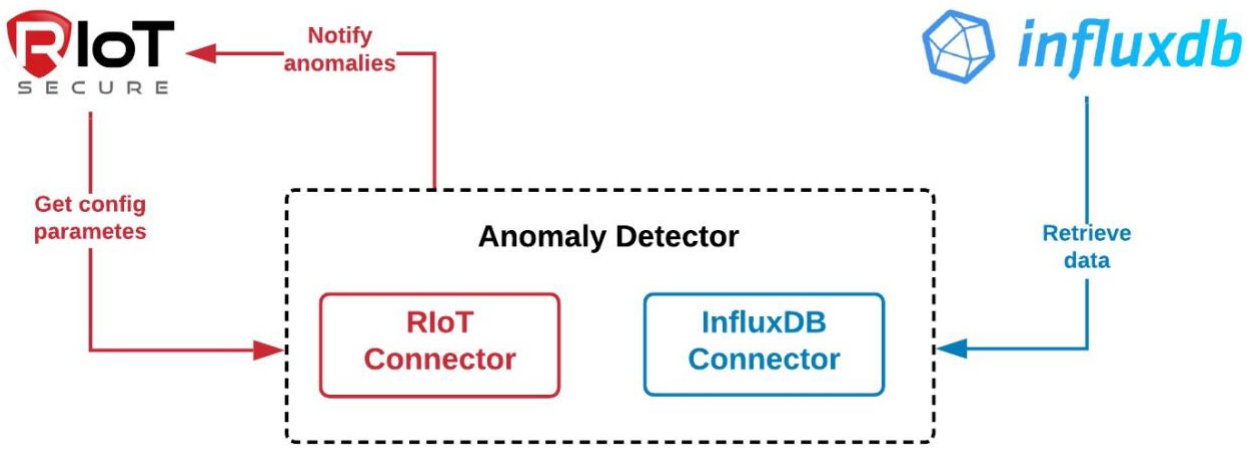
The anomaly detector needs 2 parameters:

- *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 anomaly detection techniques 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 architecture 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.

```
def create_auth(self, method):
    hash1 = '7CC384C2A10CEA62FB2A37CFDA222C04'

    now = '{:8X}'.format(int(time.time()))
    POOL = "ABCDEF0123456789"
    nonce_list = []
    for i in range(0, 24):
        nonce_list.append(POOL[random.randint(0, len(POOL) - 1)])
    nonce_str = ''.join(nonce_list)
    nonce = now + nonce_str

    auth2 = method + ':' + self._uri
    hash_object = hashlib.md5(auth2.encode())
    hash2 = hash_object.hexdigest().upper()

    auth3 = hash1 + ':' + nonce + ':' + hash2
    hash_object = hashlib.md5(auth3.encode())
    authority = hash_object.hexdigest().upper()

    return "oasis username=\"" + self._username + "\", nonce=\"" + nonce + "\", authority=\"" + authority + "\""
```

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).

```
def get_config_param(self):
    headers = {
        'Authorization': self.create_auth('GET'),
        'Content-Type': "application/json",
        'Cache-Control': "no-cache"
    }
    _uri = self._uri + "?expand"
    response = requests.get(self._base_uri + _uri, headers=headers)

    log = logging.getLogger('__main__')

    if response.status_code == 200:
        tol = response.json()['keys']['net_anomoly']['tolerance']
        freq = response.json()['keys']['net_anomoly']['frequency']
        self._update_values(tol, freq)
    elif response.status_code == 404:
        log.error('Get configuration parameters: Error 404 in connecting to RioT server')
    elif response.status_code == 401:
        log.error('Get configuration parameters: Error 401: Unauthorized access in connecting to RioT server')
    return self._tolerance, self._frequency
```

Influx Connector

```
def get_influxdb_connection(self):
    client = influxdb_client.InfluxDBClient(
        url=self._url,
        token=self._token,
        org=self._token
    )
    return client.query_api()
```

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

```
q = 'import "influxdata/influxdb/schema"\
    schema.tagValues(\
    bucket:"' + influx_connector.get_bucket() + '",\
    tag: "' + tag + '",\
    start: -' + str(riot_connector.get_frequency()) + 'h)'
org = influx_connector.get_org()
query_api = influx_connector.get_influxdb_connection()
try:
    result = query_api.query(org=org, query=q)
except Exception as err:
    logger.exception(err)
    return
else:
    results = []
    for table in result:
        for record in table.records:
            results.append((record.get_value()))
```

Dockerization

Politecnico di Milano investigated the possibility of containerizing the anomaly detection component and easing the configuration of Telegraph, InfluxDB, and the anomaly detection component adhering to the infrastructure as code principle.
The current version of the anomaly detection components satisfies all those requirements. Images of the dockerized component were uploaded in Docker Hub with all the tags that make it possible to version the component and coherently deploy it with Telegraph, InfluxDB, and the solution developed by RIoT. Detailed information regarding influxdb and telegraf on Docker Hub can be found here: https://hub.docker.com/_/influxdb, https://hub.docker.com/_/telegraf.
The following figures shows, respectively, the Dockerfile and the docker-compose.yml files.

```
FROM amancevice/pandas:1.1.4-alpine

WORKDIR /anomaly_detector

ENV FLASK_APP main.py
ENV FLASK_RUN_HOST 0.0.0.0

#Install necessary packages for running Flask and connect to Redis
RUN apk add --no-cache gcc musl-dev linux-headers

COPY . .

RUN pip install -r requirements.txt

ENTRYPOINT ["python", "main.py"]
```

```
version: "3"

services:
    anomalydetector:
        image: teaprj/anomaly_detector:0.42
        ports:
            - "5001:5000"
        command: "-uri ${URI} -username ${USERNAME} -base_uri ${BASE_URI} -bucket ${BUCKET} -org ${ORG} -token ${TOKEN} -url ${INFLUX_URL}"
    influxdb:
        image: influxdb:2.1.1
        volumes:
            - $PWD/data:/var/lib/influxdb2
        environment:
            DOCKER_INFLUXDB_INIT_MODE: setup
            DOCKER_INFLUXDB_INIT_USERNAME: admin
            DOCKER_INFLUXDB_INIT_PASSWORD: influxdb
            DOCKER_INFLUXDB_INIT_ORG: ${ORG}
            DOCKER_INFLUXDB_INIT_BUCKET: ${BUCKET}
            DOCKER_INFLUXDB_INIT_ADMIN_TOKEN: ${TOKEN}
        ports:
            - "8086:8086"

    telegraf:
        image: library/telegraf:1.21.2
        hostname: telegraf
        container_name: telegraf
        volumes:
            - ./telegraf:/etc/telegraf
            - /var/run/docker.sock:/var/run/docker.sock
```

RESTful API

Politecnico di Milano designed some REST APIs to monitor the anomaly detection component's status and provide feedback to the component about the identified anomalies. This way, users can help the anomaly detector distinguish between correctly identified anomalies and false positives. In particular, the Flask server exposes the following APIs:

GET /
It returns a text 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 transactions). 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.

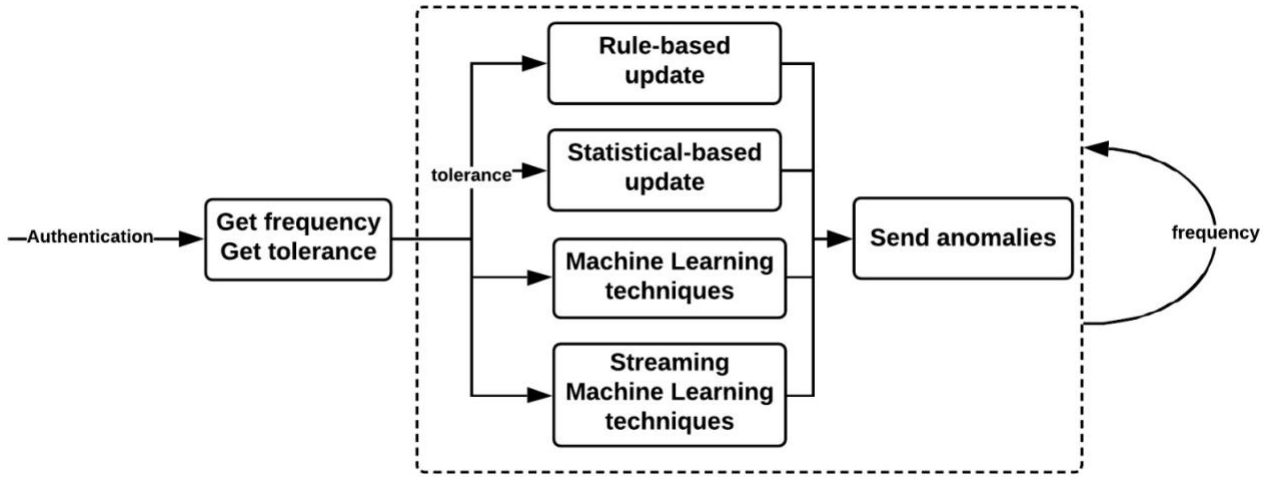
User guide

To launch the Anomaly Detector, one must execute from terminal the docker-compose file through the command *"docker-compose up -d"*. Notably, we need 7 parameters i.e., uri, username, base_uri, bucket, org, token, and url, to run the container. 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. You can set them into the .env file.

```
URI=/global
USERNAME=service
BASE_URI=https://demo.riotsecure.io:6443
BUCKET=riot
ORG=polimi
TOKEN=d2VsY29tZQ==
INFLUX_URL=http://influxdb:8086
```

The code of the anomaly_detector can be accessed here:
https://github.com/TimeEvolvingAnalytics/Anomaly-Detector

Future work



The next steps of the work will consist in deploying both the Machine Learning and Streaming Machine Learning approaches into the actual architecture. The reason why Politecnico di Milano has not yet deployed them is that it aims to achieve high precision results instead of a high recall. Since the data used to test the approaches do not contain anomalies, these approaches must be still optimized before being ready for a production environment.
Moreover, Politecnico di Milano plans to introduce another Rest API to confirm whether 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.