# INTRODUCTION

The main goal of the visualization tool is to be able to visualize the three parameters sf alloc, sf period and resource percentage which are determined by the adaptive mbsfn solution. This tool needs to meet a few requirements. First of all it needs to be able to visualize data in both a real time manner as well as in a replay manner. Secondly the data needs to be visualizable in different ways e.g. a bardiagram, piechart or table. Lastly, it needs to be easy and intuitive to use.

As mentioned before, MatplotViz only uses the python library Matplotlib and csv files to visualize and store its data and is completely development in house. Overall this tool is easy to use and meets two of the three main requirements and includes a lot of extra perks but the main drawback of MatplotViz however was the difficulty to visualize the data in a different way then a bardiagram or piechart. ThingsViz is a more mature tool which meets all the of the main requirements mentioned before and is even more practical and intuitive in use. However, the main constraint of ThingsViz is that it depends heavily on a third party data visualization platform Thingsboard. If Thingsboard would go out of business or would cancel their free-to-use version, ThingsViz would need to adapt to these changes.

# ARCHITECTURE OVERVIEW

The basic architecture of ThingsViz is as follows:

## CLIENT

First of all a data generating client will generate the three desired parameters. This client can either be a random generator or a “real life” client. The random generator can be started by running the the generators/client.py file which randomly generates the values of the parameters. This generator can by used for simple demonstration and testing/development purposes. The “real life” client will be the adaptive mbsfn solution, which calculates every 0.64 seconds new parameter values. Once the parameter values are calculated/generated, the client will connect to the ThingsViz server which is listening for incoming messages on a socket located at a certain port of choice and send the parameters in a comma separated string form.

## SERVER

The ThingsViz server is a simple python server which opens up a TCP socket on the server machine and listens for incoming messages. By default the server is ran on the same “localhost” machine on which the adaptive mbsf EnodeB is running but in theory this can be any desired machine. However this is in not recommended due to the introduction of extra undesired latencies. When a new message, containing the parameters, is received by the server it will open up a connection to the MQTT broker (more information about MQTT later) and publish this parameters in a JSON string format to the broker. Note that the ThingsViz server doubles up as a receiver on the socket communication side as well as a sender on the MQTT communication side.

## MQTT BROKER/ THINGSBOARD PLATFORM

As mentioned before the ThingsViz server publishes the parameters to a MQTT broker. But first of all, what is MQTT exactly? MQTT is an application layer protocol which uses the underlying TCP protocol and was developed to be mainly used in Internet of Things use cases. MQTT communication needs to meet two main requirements. First of all, it needs to be fast and can’t introduce a lot of latencies because of the real time characteristics of a lot of IoT applications. Secondly it needs to have a very simple client side because it needs to be able to be implemented in very small, controllers and IoT devices. MQTT uses a publish/subscribe architecture. A centralized broker will keep track of all the connections with clients as well as store and forward incoming messages and will be responsible for all the communications logics. Clients simply need to know the address and port of the broker in order to be able to send messages. A client will publish messages on a topic of choice. If another client wants to receives the messages of a certain topic, it simply needs to communicate this to the broker via a subscribe-request. The broker in turn will receive these subscribe-request and it will make sure that all the messages that are send on the topic of interest will be forwarded to the subscribed client(s). One major constraint of MQTT is the fact that the sender as well as the receiver(s) need to know the exact name of the topic on which to subscribe and publish.

ThingsViz uses the MQTT protocol to transport the messages from the server to the Thingsboard platform. The broker is ran and managed by Thingsboard and needs zero configuration. The broker will always inherit the address of the Thingsboard platform. E.g. if Thingsboard is ran locally, the address of the broker will simply be localhost, if Thingsboard is ran in the cloud the address will be “thingsboard.cloud”. For optimal real time performance it is recommended to run Thingsboard on the same machine as the EnodeB or on the same Local Area Network. The Thingsboard platform will implicitly subscribe on the topics on which the ThingsViz server is publishing. This means that all the data can easily be visualized by the configurable Thingsboard dashboards. However, not only does Thingsboard receive and visualizes the messages but it also stores the data by default into a local postgresql database. This needs no configuration and a user can seamlessly visualize the data from for example the past week without being hassled by complicated database queries.

A few remarks: As mentioned before, both the ThingsViz server as well as the Thingsboard platform can be ran on the same localhost machine on which the EnodeB is running. However it is possible to run both of them on other machines if for example the EnodeB hasn’t got enough compute power to handle everything or if it is not possible to hook up a monitor to the EnodeB. Always make sure that the network\_components/client.py script has the port and address of the ThingsViz server configured correctly and that the ThingsViz server has the address and port of the broker configured correctly. By default everything is set to “localhost” and assumed to be ran on one machine.

# INSTALLATION

## CLIENT

**Pre requirements**: Socket libraries in python as well as in c++ are usually included by default.

* To install the random generator simply download the CLIENTSIDE/generators directory. This directory contains a client.cpp/client.py which will generate random parameters and send them to a server socket.
* To install the modified adaptive mbsfn solution, first download the CLIENTSIDE/adaptive\_mbsfn/mac directory. Replace the original mac directory in the original adaptive\_mbsfn project with the new mac directory. Rerun “cmake ../”, “sudo make”, “sudo make install” and “sudo ldconfig”. From now on the adaptive\_mbsfn module will not only print the parameters on the terminal, just as before, but it will also try to send the parameters to the server if possible. Note that the solution can still be ran without running a server. It will just print “connection failed” on the terminal alongside the parameters.

Modified files:

1. Network\_components/client.cc ADDED
2. Mac.cc MODIFIED Added line 907-911
3. CMakeLists.txt MODIFIED Set(…, net\_comp/client.cc)

## SERVER

**Pre requirements**: Install the paho-mqtt client: “pip install paho-mqtt”

* To install the ThingsViz python server, simply download the SERVERSIDE/ThingsViz directory. The serverside code can be found in the src directory.

## THINGSBOARD

* ThingsBoard can be installed both natively on Ubuntu or on Docker for Ubuntu.
* <https://thingsboard.io/docs/user-guide/install/ubuntu/>
* <https://thingsboard.io/docs/user-guide/install/docker/>
* ThingsBoard can be ran in the cloud => NO INSTALLATION REQUIRED
* **Pre requirements** **for Thingsboard on docker:**
* <https://docs.docker.com/engine/install/ubuntu/>
* <https://docs.docker.com/compose/install/>
* TIP: install portainer as a useful docker manager UI for linux
* Remarks for installing on Docker
* Run “sudo docker-compose pull”
* Run “sudo docker-compuse up”

HOW TO SETUP DASHBOARD AND THE MQTT BROKER IN THINGSBOARD:

1. Import the Device profile .json configuration file from the SERVERSIDE/ThingsViz/thingsboard\_templates/profiles directory

IF IMPORT DOESN’T WORK RECREATE THE SCREENSHOT

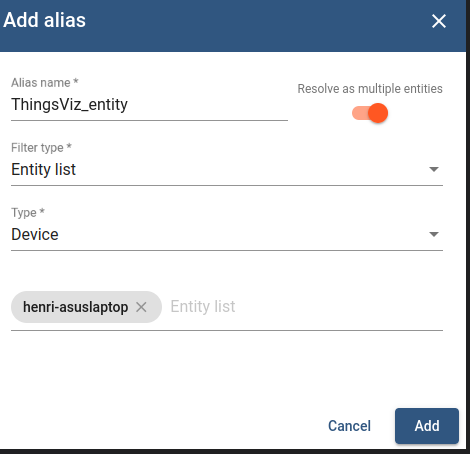
Afbeelding met tekst

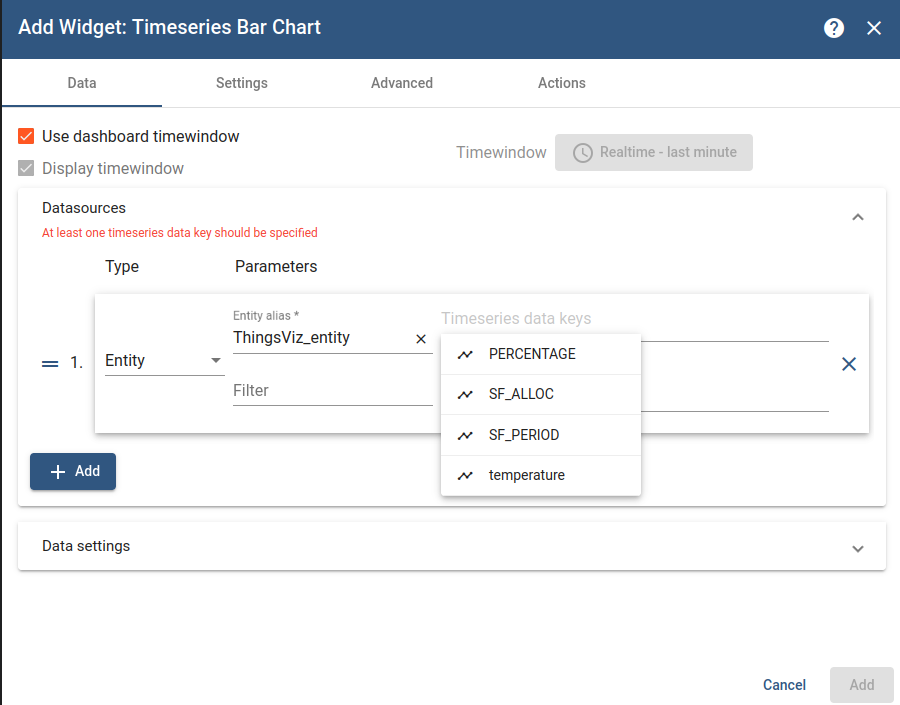
Automatisch gegenereerde beschrijving

1. Add a new device with the device name set to the hostname of the machine on which Thingsboard will be ran. Set the Device profile to ThingsViz

Afbeelding met tafel

Automatisch gegenereerde beschrijving

1. Import the dashboards .json configuration files from the directory SERVERSIDE/ThingsViz/thingsboard\_templates/dashboards or recreate them. REMARK: Make sure that a entity alias is added which references the device on which Thingsboard is ran. When adding new widgets on the dashboard follow the screenshot to correctly add the data source from which the widget needs to pull data. 



When all steps are followed, you’re ready to run the entire pipeline.

THINGSBOARD CLOUD

Login: [henri.declercq@ugent.be](mailto:henri.declercq@ugent.be)

Pwd: 4\*\*\*\*\*\*\*3\*\*\*\*\*\*\*\*\*

addressOfBroker: thingsboard.cloud

portOfBroker: 1883

* ADD NEW DEVICE IN DEVICE GROUP ThingsViZ
* NEW DASHBOARD => NEW ALIAS OF ENTITY GROUP THINGSVIZ DEVICES

OR RUN THINGSBOARD LOCALLY WITH DOCKER OR ON DEVICE

1. Pipeline :

* C++ client adaptive sends the parameters to the python server and this server forwards the parameters to the thingsboard.cloud broker ExPLAIN DOCKER INSTALLATION AND PORTAINER AS WELL HANDY GUI AND METION DOCKER COMPOSE ISNTALL AS WELL

<https://vitux.com/how-to-install-portainer-dcoker-manager-in-ubuntu-20-04/>

<https://thingsboard.io/docs/user-guide/install/docker/>

remaks: run SUDO docker pull and SUDO docker-compose up

portainer pswd: urbis

portainer username: adminaturbis

portainer port: 9000

1. **HOW TO RUN THE PIPELINE**