NILMTK Rapid Experimentation API

A API torna a execução de experimentos extremamente rápida e eficiente, com ênfase na criação de experimentos reproduzíveis com ajuste fino, onde o desempenho do modelo e dos parâmetros pode ser facilmente avaliado em um relance.

Importando bibliotecas

In [1]:

```
from matplotlib import rcParams
import matplotlib.pyplot as plt
import pandas as pd
import nilmtk
from nilmtk import MeterGroup
from nilmtk.api import API
import warnings
warnings.filterwarnings("ignore")

plt.style.use('ggplot')
rcParams['figure.figsize'] = (13, 10)

import pathlib
pathlib.Path().resolve()
```

Out[1]:

```
PosixPath('/home/hb/projetos/nilmtk')
```

Convertendo a base de dados

In [59]:

```
from nilmtk.dataset_converters import convert_redd
convert_redd('./BD/CASA/', './data/teste10.h5')
```

```
Loading house 1... 1 2 3
Loaded metadata
Done converting YAML metadata to HDF5!
Done converting REDD to HDF5!
```

Importando a base de dados

In [2]:

```
from nilmtk import DataSet
from nilmtk.utils import print_dict

redd = DataSet('./data/teste10.h5')
#iawe = DataSet('/data/iawe.h5')

print_dict(redd.metadata)
print_dict(redd.buildings)
```

- **name**: REDD
- **long_name**: The Reference Energy Disaggregation Data set
- **creators**:
  - Kolter, Zico
  - Johnson, Matthew
- **publication_date**: 2011
- **institution**: Massachusetts Institute of Technology (MIT)
- **contact**: zkolter@cs.cmu.edu
- **description**: Several weeks of power data for 6 different homes.
- **subject**: Disaggregated power demand from domestic buildings.
- **number_of_buildings**: 1
- **timezone**: US/Eastern
- **geo_location**:
  - **locality**: Massachusetts
  - **country**: US
  - **latitude**: 42.360091
  - **longitude**: -71.09416
- **related_documents**:
  - http://redd.csail.mit.edu (http://redd.csail.mit.edu)
  - J. Zico Kolter and Matthew J. Johnson. REDD: A public data set for energy disaggregation research. In proceedings of the SustKDD workshop on Data Mining Applications in Sustainability, 2011. http://redd.csail.mit.edu/kolter-kddsust11.pdf (http://redd.csail.mit.edu/kolter-kddsust11.pdf)
- **schema**: https://github.com/nilmtk/nilm_metadata/tree/v0.2 (https://github.com/nilmtk/nilm_metadata/tree/v0.2)
- **meter_devices**:
  - **eMonitor**:
    - **model**: eMonitor
    - **manufacturer**: Powerhouse Dynamics
    - **manufacturer_url**: http://powerhousedynamics.com (http://powerhousedynamics.com)
    - **description**: Measures circuit-level power demand. Comes with 24 CTs. This FAQ page suggests the eMonitor measures real (active) power: http://www.energycircle.com/node/14103 (http://www.energycircle.com/node/14103) although the REDD readme.txt says all channels record apparent power.
    - **sample_period**: 5
    - **max_sample_period**: 30
    - **measurements**:
      - {'physical_quantity': 'power', 'type': 'active', 'upper_limit': 1142, 'lower_limit': 0}
      - {'physical_quantity': 'power', 'type': 'apparent', 'upper_limit': 1215, 'lower_limit': 0}

    - {'physical_quantity': 'power', 'type': 'reactive', 'upper_limit': 901, 'lower_limit': 0}
    - {'physical_quantity': 'power factor', 'upper_limit': 1, 'lower_limit': 0}
    - {'physical_quantity': 'voltage', 'upper_limit': 232, 'lower_limit': 0}
    - {'physical_quantity': 'current', 'upper_limit': 6, 'lower_limit': 0}
  - **wireless**: False
- **REDD_whole_house**:
  - **description**: REDD's DIY power meter used to measure whole-home AC waveforms at high frequency. To quote from their paper: "CTs from TED (http://www.theenergydetective.com (http://www.theenergydetective.com)) to measure current in the power mains, a Pico TA041 oscilloscope probe (http://www.picotechnologies.com (http://www.picotechnologies.com)) to measure voltage for one of the two phases in the home, and a National Instruments NI-9239 analog to digital converter to transform both these analog signals to digital readings. This A/D converter has 24 bit resolution with noise of approximately 70 µV, which determines the noise level of our current and voltage readings: the TED CTs are rated for 200 amp circuits and a maximum of 3 volts, so we are able to differentiate between currents of approximately ((200))(70 × 10−6)/(3) = 4.66mA, corresponding to power changes of about 0.5 watts. Similarly, since we use a 1:100 voltage stepdown in the oscilloscope probe, we can detect voltage differences of about 7mV."
  - **sample_period**: 0.5
  - **max_sample_period**: 30
  - **measurements**:
    - {'physical_quantity': 'voltage', 'upper_limit': 230, 'lower_limit': 0}
    - {'physical_quantity': 'current', 'upper_limit': 15, 'lower_limit': 0}
    - {'physical_quantity': 'power', 'type': 'active', 'upper_limit': 3016, 'lower_limit': 0}
    - {'physical_quantity': 'frequency', 'upper_limit': 61, 'lower_limit': 0}
    - {'physical_quantity': 'power factor', 'upper_limit': 1, 'lower_limit': 0}
  - **wireless**: False

- **1**: Building(instance=1, dataset='REDD')

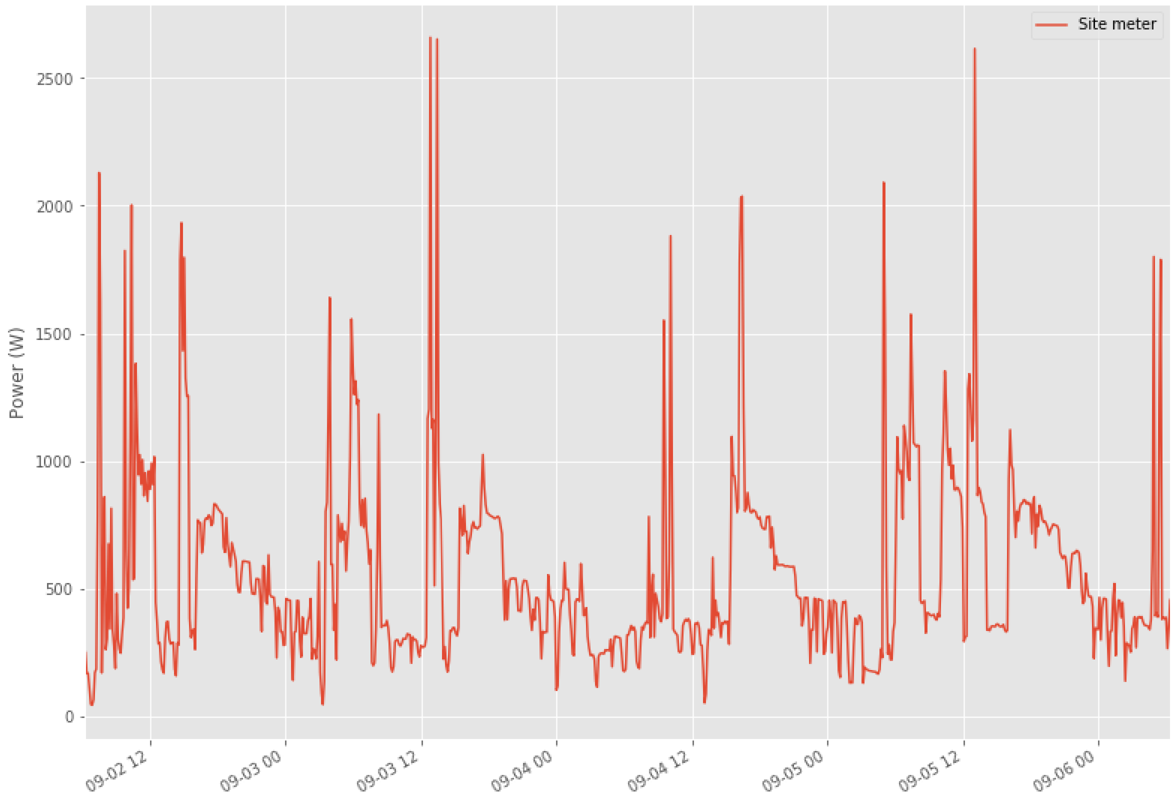
Carregando exemplo de uma casa/eletrodoméstico

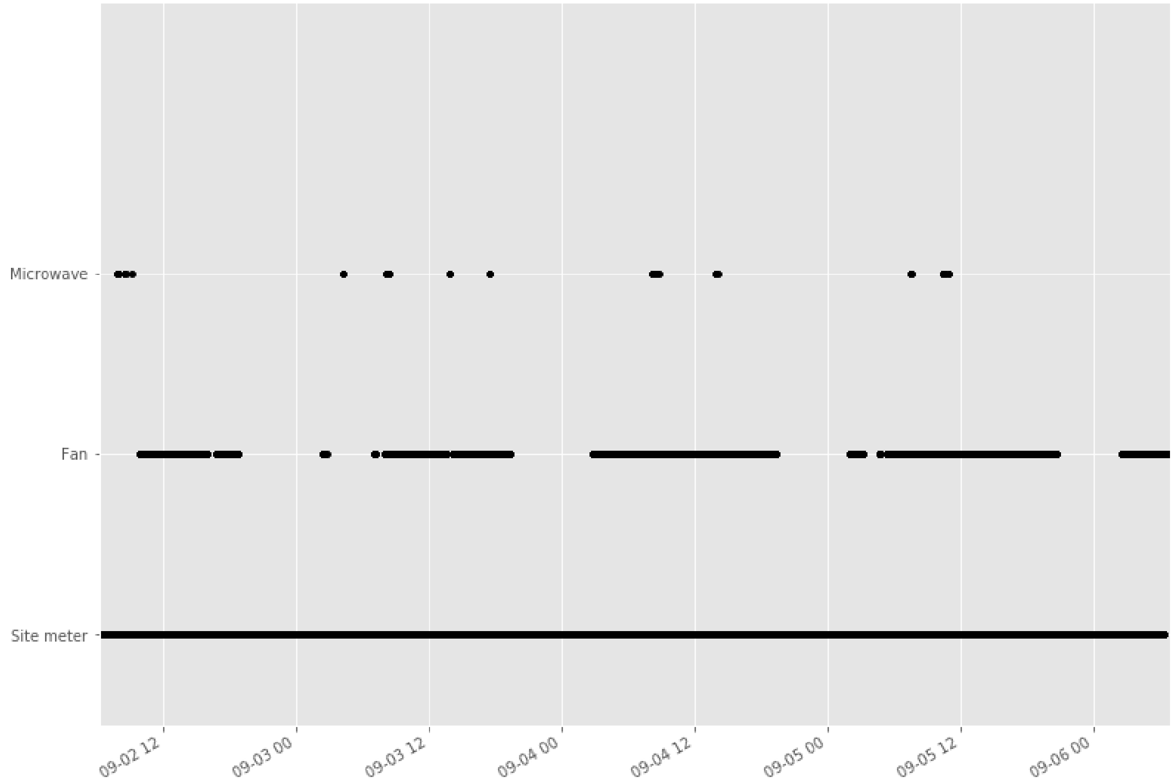
In [3]:

```
build = 1
elec = redd.buildings[build].elec
elec.mains().power_series_all_data().head()
elec.mains().plot()
# sns.set_palette("Set3", n_colors=12)
# Set a threshold to remove residual power noise when devices are off
elec.plot_when_on(on_power_threshold = 40) # Plot appliances when they are in u
```

Out[3]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f129d5356d0>
```

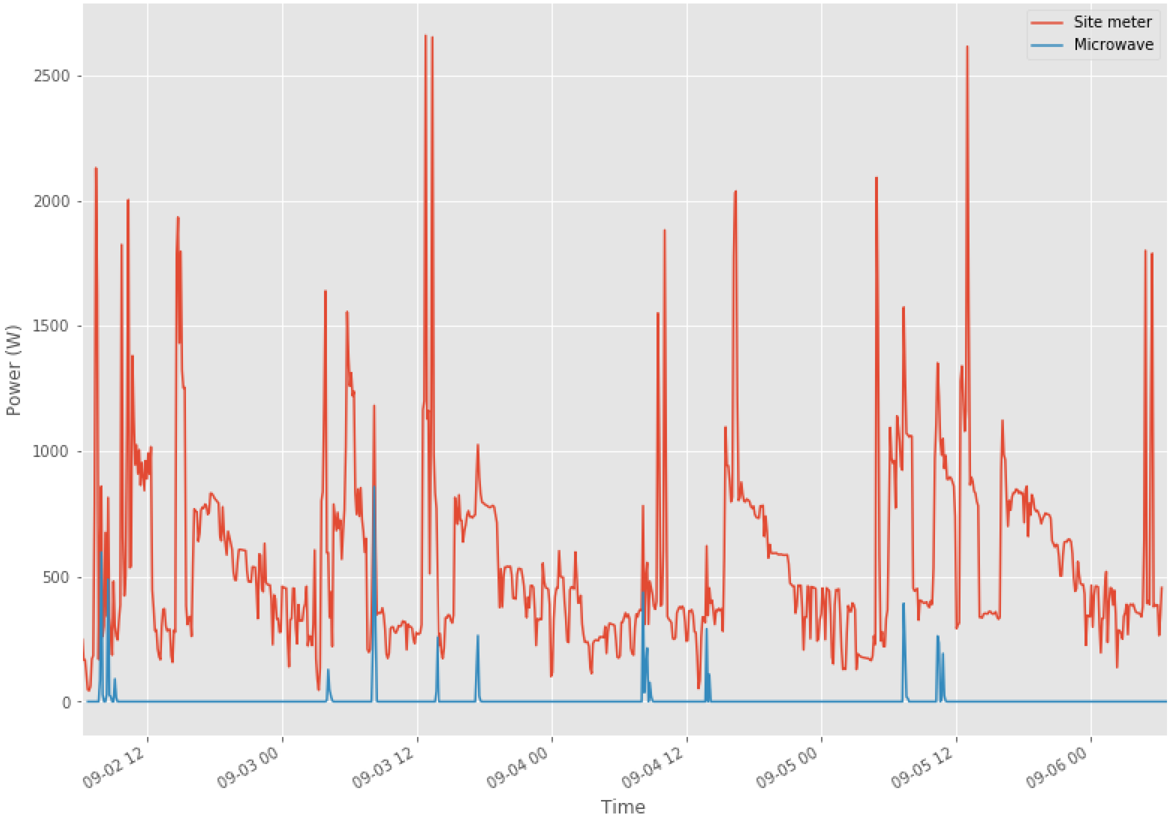



Microwave
Fan
Site meter
09-02 12
09-03 00
09-03 12
09-04 00
09-04 12
09-05 00
09-05 12
09-06 00

In [4]:

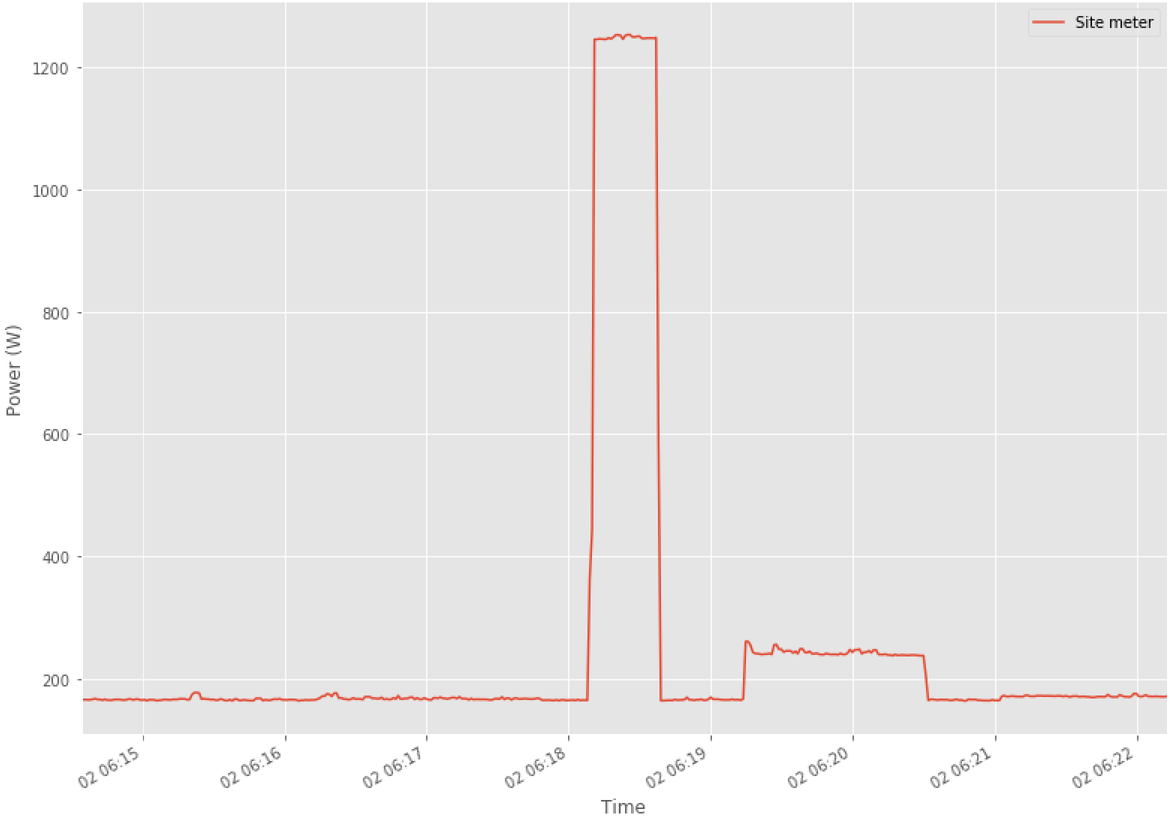
```
microwave = elec['microwave']
microwave.available_columns()
print(next(microwave.load()).head())

from nilmtk.elecmeter import ElecMeterID

meter1 = elec[ElecMeterID(instance=0, building=build, dataset='REDD')]

redd.set_window(start='2021-09-02 06:14:34', end='2021-09-06 10:24:14')
meter1.plot()
elec['microwave'].plot()
plt.xlabel("Time");
```

```
physical_quantity          power
type                      active
2021-09-02 06:47:51-04:00    0.0
2021-09-02 06:47:56-04:00    0.0
2021-09-02 06:48:01-04:00    0.0
2021-09-02 06:48:06-04:00    0.0
2021-09-02 06:48:11-04:00    0.0
```



In [70]:

```
redd.set_window(start='2021-09-02 06:14:34', end='2021-09-02 06:22:14')
meter1.plot() # 1 segundo
elec['microwave'].plot() # 3 segundos
plt.xlabel("Time");
```



In [71]:

```
next(elec.load())

```

```
Loading data for meter ElecMeterID(instance=3, building=1, dataset='RE
DD')
Done loading data all meters for this chunk.
```

Out[71]:

physical_quantity	**frequency**	**voltage**	**power**		**current**	**power**	**power factor**
type			**reactive**	**apparent**		**active**	
2021-09-02 06:14:30-04:00	NaN	NaN	NaN	NaN	NaN	167.199997	NaN
2021-09-02 06:14:35-04:00	NaN	NaN	NaN	NaN	NaN	167.259995	NaN
2021-09-02 06:14:40-04:00	NaN	NaN	NaN	NaN	NaN	167.559998	NaN
2021-09-02 06:14:45-04:00	NaN	NaN	NaN	NaN	NaN	166.839996	NaN
2021-09-02 06:14:50-04:00	NaN	NaN	NaN	NaN	NaN	167.160004	NaN
...	...	...	...	...	...	...	...
2021-09-02 06:21:50-04:00	NaN	NaN	NaN	NaN	NaN	172.639999	NaN
2021-09-02 06:21:55-04:00	NaN	NaN	NaN	NaN	NaN	172.979996	NaN
2021-09-02 06:22:00-04:00	NaN	NaN	NaN	NaN	NaN	173.940002	NaN
2021-09-02 06:22:05-04:00	NaN	NaN	NaN	NaN	NaN	172.400009	NaN
2021-09-02 06:22:10-04:00	NaN	NaN	NaN	NaN	NaN	172.199997	NaN

93 rows × 7 columns

In [5]:

```
next(meter1.load())

```

Out[5]:

physical_quantity	power
type	**active**
2021-09-02 06:14:34-04:00	167.199997
2021-09-02 06:14:35-04:00	167.199997
2021-09-02 06:14:35-04:00	167.199997
2021-09-02 06:14:36-04:00	167.199997
2021-09-02 06:14:36-04:00	166.899994
...	...
2021-09-06 06:24:13-04:00	458.399994
2021-09-06 06:24:14-04:00	458.399994
2021-09-06 06:24:14-04:00	458.299988
2021-09-06 06:24:15-04:00	458.299988
2021-09-06 06:24:15-04:00	459.000000

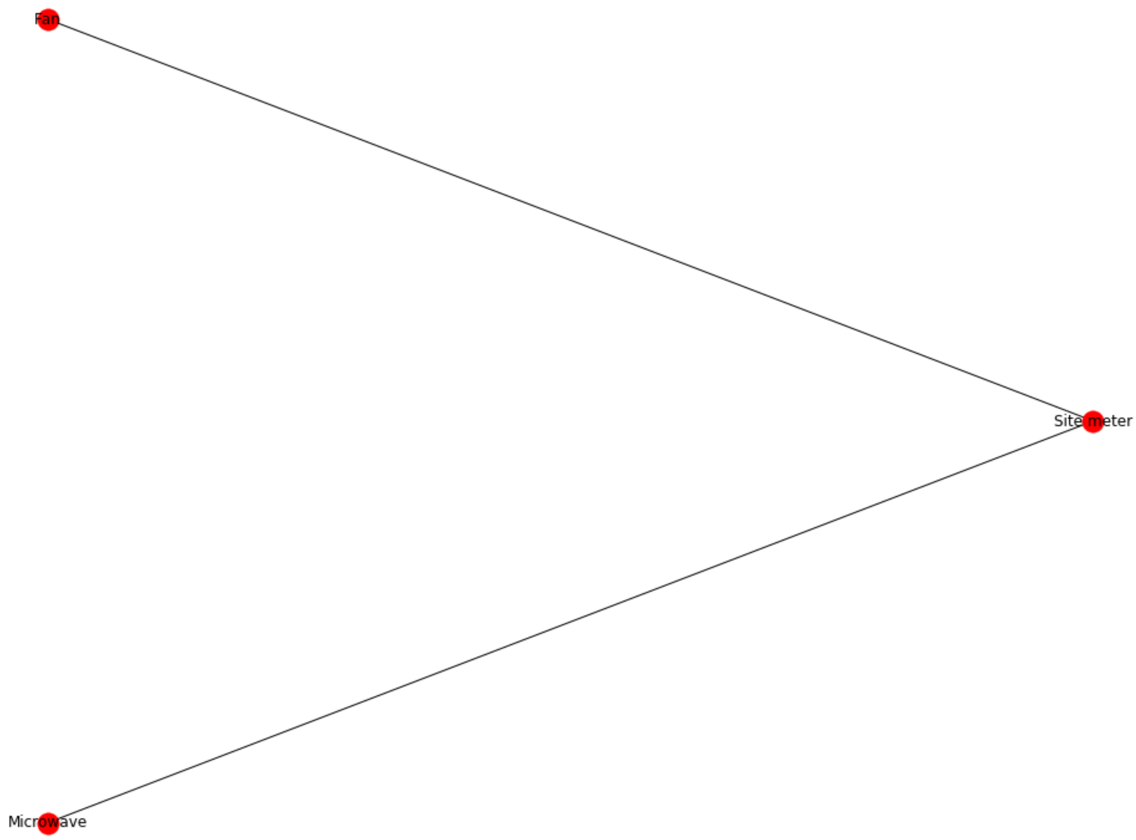
692159 rows × 1 columns

In [66]:

```
elec.draw_wiring_graph()

```

Out[66]:

```
(<networkx.classes.digraph.DiGraph at 0x7effebc18090>,
 <matplotlib.axes._axes.Axes at 0x7effebc22490>)
```



Importamos os algoritmos que desejamos executar os experimentos:

- Mean: Mean Algorithm
- Hart's Algorithm
- CO: Combinatorial Optimization
- Discriminative Sparse Coding

- Additive Factorial Hidden Markov Model
- Additive Factorial Hidden Markov Model with Signal Aggregate Constraints
- DSC: Discriminative Sparse Coding
- RNN: Long short-term memory - LSTM
- DAE: Denoising Auto Encoder
- Seq2Point*
- Seq2Seq
- WindowGRU/Online GRU: Similar a LSTM, mas usa Gated Recurrent Unit (GRU)
- ELM

In [4]:

```
from nilmtk.disaggregate import Mean,CO,Hart85
# from nilmtk_contrib.disaggregate import AFHMM,AFHMM_SAC,DSC,RNN,Seq2Point,Seq
from nilmtk_contrib.disaggregate import RNN,Seq2Point


```

```
Using TensorFlow backend.
```

Em seguida, inserimos os valores para os diferentes parâmetros no dicionário. Como precisamos de vários aparelhos, inserimos os nomes de todos os aparelhos necessários no parâmetro *'appliances'*.

Métricas: https://github.com/nilmtk/nilmtk/blob/master/nilmtk/losses.py (https://github.com/nilmtk/nilmtk/blob/master/nilmtk/losses.py)

Error: https://github.com/nilmtk/nilmtk-contrib/issues/56 (https://github.com/nilmtk/nilmtk-contrib/issues/56)

In [8]:

```
experiment1 = {
  'power': {'mains': ['active'],'appliance': ['active']},
  'sample_rate': 5,
  'display_predictions': True,
  'artificial_aggregate': False,
  'DROP_ALL_NANS': True,
  'site_only': False,
  #'chunksize': 20000,
  'appliances': ['microwave', 'fan'],
  'methods': {
      'Mean':Mean({}),
      "CO":CO({}),
      'Hart85':Hart85({}),
      'RNN':RNN({'n_epochs':50,'batch_size':1024}),
#       'Seq2Seq':Seq2Seq({'n_epochs':5,'batch_size':32})
      'Seq2Point':Seq2Point({'n_epochs':50,'batch_size':1024})
      #"DSC":{'learning_rate':5*1e-10,'iterations':100}
      #"AFHMM":AFHMM({}),
      #"AFHMM_SAC":AFHMM_SAC({}),
      #FHMM_EXACT({'num_of_states':2})
      #'Convlstm': Convlstm({'n_epochs':30,}),
  },
  'train': {
    'datasets': {
        'Redd': {
            'path': './data/teste10.h5',
            'buildings': {
                1: {
                    'start_time': '2021-09-02',
                    'end_time': '2021-09-04'
                    }
                }
            }
        }
    },
  'test': {
    'datasets': {
        'Redd': {
            'path': './data/teste10.h5',
            'buildings': {
                1: {
                    'start_time': '2021-09-05',
                    'end_time': '2021-09-06'
                    }
                }
            }
        },
        'metrics':['rmse', 'mae', 'relative_error', 'r2score', 'nde', 'nep', 'f
    }
}
```

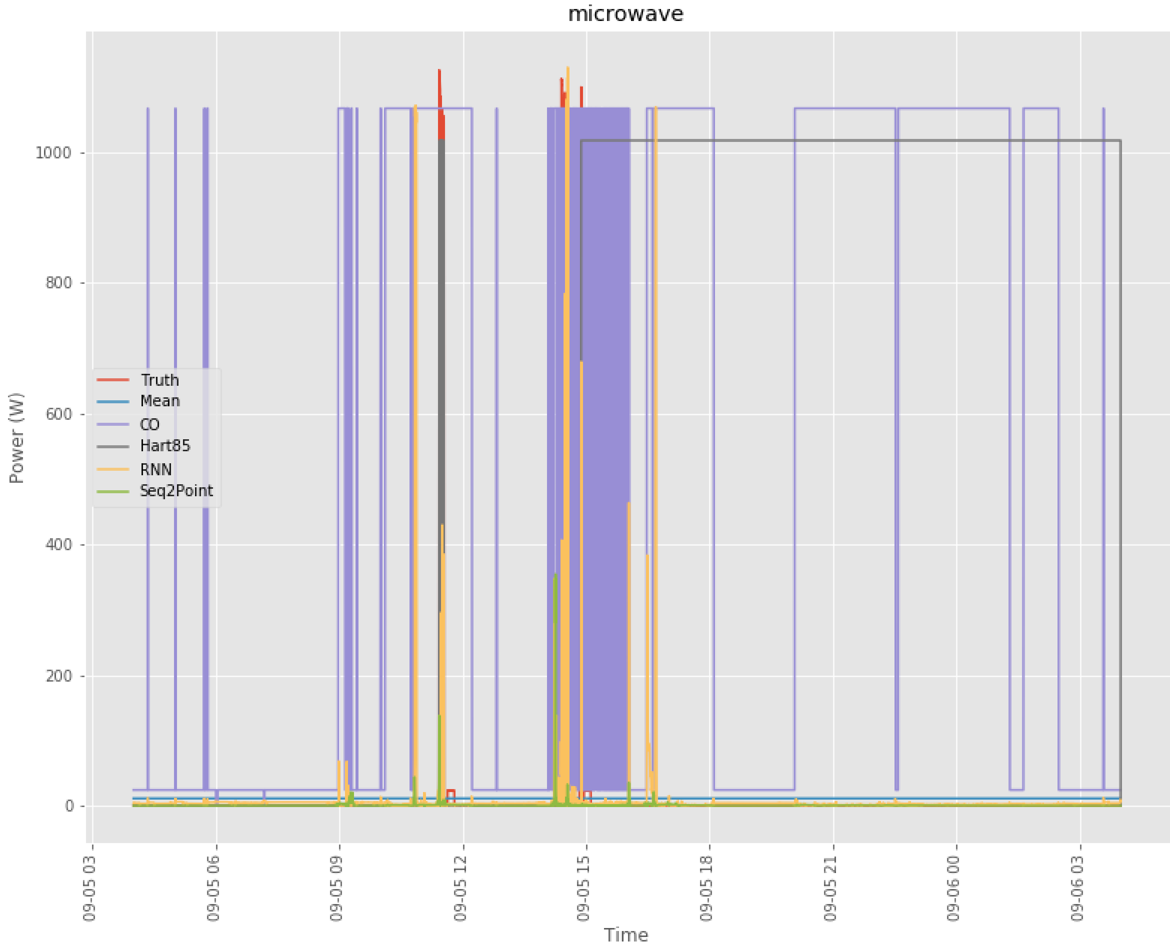
In this example experimental setup, we have set the *sample rate* at 60Hz and use Combinatorial Optimisation to disaggregate the required appliances from building 10 in the dataport dataset with the *RMSE* metric to measure the accuracy. We also specify the dates for training and testing

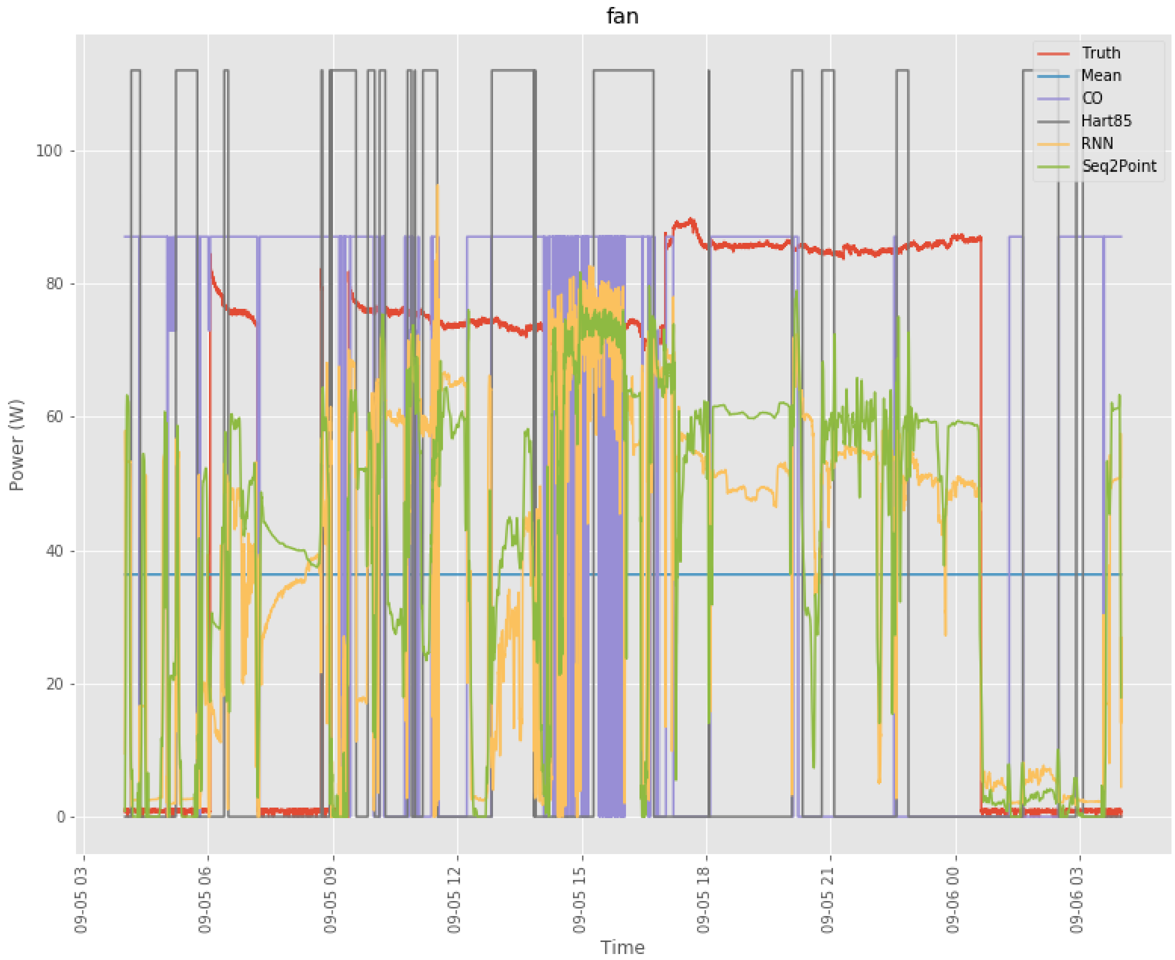
Next we provide this experiment dictionary as input to the API.

In [9]:

```
api_results_experiment_1 = API(experiment1)
```

```
Joint Testing for all algorithms
Loading data for  Redd  dataset
Dropping missing values
Generating predictions for : Mean
Generating predictions for : CO
...............CO disaggregate_chunk running.............
Generating predictions for : Hart85ave'
Finding Edges, please wait ...
Edge detection complete.
Creating transition frame ...
Transition frame created.
Creating states frame ...
States frame created.
Finished.
Generating predictions for : RNN
Generating predictions for : Seq2Point
............  rmse  ..............
                Mean          CO      Hart85        RNN  Seq2Point
microwave  81.208117  710.920948  749.999958  93.682481  81.004409
fan        41.437562   63.119786   68.025881  32.959566  31.548666
............  mae  ..............
                Mean          CO      Hart85        RNN  Seq2Point
microwave  17.200323  486.960693  553.253418  13.115963   7.302773
fan        40.981853   50.396896   57.150852  26.339155  25.144955
............  relative_error  ..............
               Mean         CO     Hart85       RNN  Seq2Point
microwave  1.422261   0.963681   3.409196  2.780659   2.410225
fan        1.098067  25.810015  41.241970  1.789826   1.920159
............  r2score  ..............
               Mean         CO     Hart85       RNN  Seq2Point
microwave -0.002878 -75.858507 -84.540510 -0.334646   0.002147
fan       -0.260763  -1.925338  -2.397765  0.202358   0.269186
............  nde  ..............
               Mean        CO    Hart85       RNN  Seq2Point
microwave  0.997994  8.736746  9.217001  1.151295   0.995490
fan        0.624317  0.950991  1.024909  0.496584   0.475326
............  nep  ..............
               Mean         CO     Hart85       RNN  Seq2Point
microwave  2.550806  72.216209  82.047409  1.945096   1.083000
fan        0.742869   0.913534   1.035961  0.477444   0.455797
............  f1score  ..............
               Mean        CO    Hart85       RNN  Seq2Point
microwave  0.053929  0.054029  0.034533  0.244373   0.054237
fan        0.815107  0.576746  0.382349  0.879807   0.890314
```


microwave
Truth
Mean
CO
Hart85
RNN
Seq2Point
Power (W)
0
200
400
600
800
1000
09-05 03
09-05 06
09-05 09
09-05 12
09-05 15
09-05 18
09-05 21
09-06 00
09-06 03
Time



raiz do erro quadrático médio (RMSE) e o erro médio absoluto (MAE)

Quanto menor o seu valor, melhor é o modelo, já que a previsão se mostra mais próxima ao valor real. Comparando as duas métricas têm se que o RMSE penaliza desvios grandes, enquanto o MAE tem pesos iguais para todos os desvios.

We can observe the prediction vs. truth graphs in the above cell. The accuracy metrics can be accessed using the following commands:

In [10]:

```python
errors_keys = api_results_experiment_1.errors_keys
errors = api_results_experiment_1.errors
for i in range(len(errors)):
    print (errors_keys[i])
    print (errors[i])
    print ("\n\n")
```

```
Redd_1_rmse
                Mean          CO      Hart85        RNN  Seq2Point
microwave  81.208117  710.920948  749.999958  93.682481  81.004409
fan        41.437562   63.119786   68.025881  32.959566  31.548666



Redd_1_mae
                Mean          CO      Hart85        RNN  Seq2Point
microwave  17.200323  486.960693  553.253418  13.115963   7.302773
fan        40.981853   50.396896   57.150852  26.339155  25.144955



Redd_1_relative_error
               Mean         CO     Hart85       RNN  Seq2Point
microwave  1.422261   0.963681   3.409196  2.780659   2.410225
fan        1.098067  25.810015  41.241970  1.789826   1.920159



Redd_1_r2score
               Mean         CO     Hart85       RNN  Seq2Point
microwave -0.002878 -75.858507 -84.540510 -0.334646   0.002147
fan       -0.260763  -1.925338  -2.397765  0.202358   0.269186



Redd_1_nde
               Mean        CO    Hart85       RNN  Seq2Point
microwave  0.997994  8.736746  9.217001  1.151295   0.995490
fan        0.624317  0.950991  1.024909  0.496584   0.475326



Redd_1_nep
               Mean         CO     Hart85       RNN  Seq2Point
microwave  2.550806  72.216209  82.047409  1.945096   1.083000
fan        0.742869   0.913534   1.035961  0.477444   0.455797



Redd_1_f1score
               Mean        CO    Hart85       RNN  Seq2Point
microwave  0.053929  0.054029  0.034533  0.244373   0.054237
fan        0.815107  0.576746  0.382349  0.879807   0.890314
```

In [11]:

```
import numpy as np
import pandas as pd

vals = np.concatenate([np.expand_dims(df.values,axis=2) for df in api_results_e


cols = api_results_experiment_1.errors[0].columns
indexes = api_results_experiment_1.errors[0].index


mean = np.mean(vals,axis=2)
std = np.std(vals,axis=2)
print ('\n\n')
print ("Mean")
print (pd.DataFrame(mean,index=indexes,columns=cols))
print ('\n\n')
print ("Standard Deviation")
print (pd.DataFrame(std,index=indexes,columns=cols))
```

```
Mean
                Mean          CO      Hart85        RNN  Seq2Point
microwave  14.775793  171.999114  187.631572  16.083603  13.264612
fan        12.205573   19.977519   23.780594   9.020677   8.672057



Standard Deviation
                Mean          CO      Hart85        RNN  Seq2Point
microwave  27.702164  279.412752  301.454921  31.961299  27.752404
fan        18.348342   25.077540   28.395279  13.174482  12.570605
```

In [ ]:

```

```