controller_usage_demonstration

June 12, 2017

This Jupyter Notebook demonstrates how to enable the GridBallast controller for a load (a water heater or a zip load) in GridLAB-D by feeding the simulator a stored grid frequency time series contained in an external file.

To run this notebook, please make sure you are in a UNIX based environment and have all the necessary python packages installed (plotly, matplotlib, numpy, pandas).

```
In [1]: !ls
```

```
controller_usage_demonstration.ipynb smSingle_base.glm
controller_usage_demonstration.pdf   smSingle_lenient_freq.glm
correct_path.sh                      smSingle_lenient_freq_lock_mode.glm
frequency.PLAYER                     smSingle_strict_freq.glm
hot_water_demand.glm                 smSingle_strict_freq_jitter300.glm
local_gd                             smSingle_strict_freq_jitter60.glm
lock_mode_schedule.glm               smSingle_strict_freq_jitter600.glm
smSingle.glm
```

The gridlab-d binary file is stored within **local_gd** directory along with libraries. We can check the version of the gridlabd using the following command:

```
In [2]: !local_gd/bin/gridlabd --version
```

```
GridLAB-D 4.0.0-17329 (feature/730:17329) 64-bit MACOSX RELEASE
```

The above listed **local_gd/bin/gridlabd** is the binary version of the gridlab-d software with controlling functionality. In addition to that, we have **.glm** files and generated **.csv** files. We also have a **frequency.PLAYER** containing the 1-second resolution frequency information.

The version of the gridlab-d binary file and the content of the frequency.PLAYER can be seen below.

If the version of the gridlab-d does not work, we can disable the comments below and run the command to compile the source and install the gridlab-d to the machine.

```
In [3]: # %%bash
        # cd ~
        # git clone -b feature/730 https://github.com/jingkungao/gridlab-d.git
        # cd gridlab-d
```

```
# cd third_party
# chmod +x install_xercesc
# . install_xercesc
# tar -xvf cppunit-1.12.0.tar.gz
# cd cppunit-1.12.0
# ./configure LDFLAGS="-ldl"
# make
# sudo make install
# cd ../..
# autoreconf -isf
# ./configure
# make
# sudo make install
```

In [4]:
```
!head -5 frequency.PLAYER
```

```
2012-01-01 00:00:00 EST,59.9769
2012-01-01 00:00:01 EST,59.9763
2012-01-01 00:00:02 EST,59.9715
2012-01-01 00:00:03 EST,59.9714
2012-01-01 00:00:04 EST,59.972
```

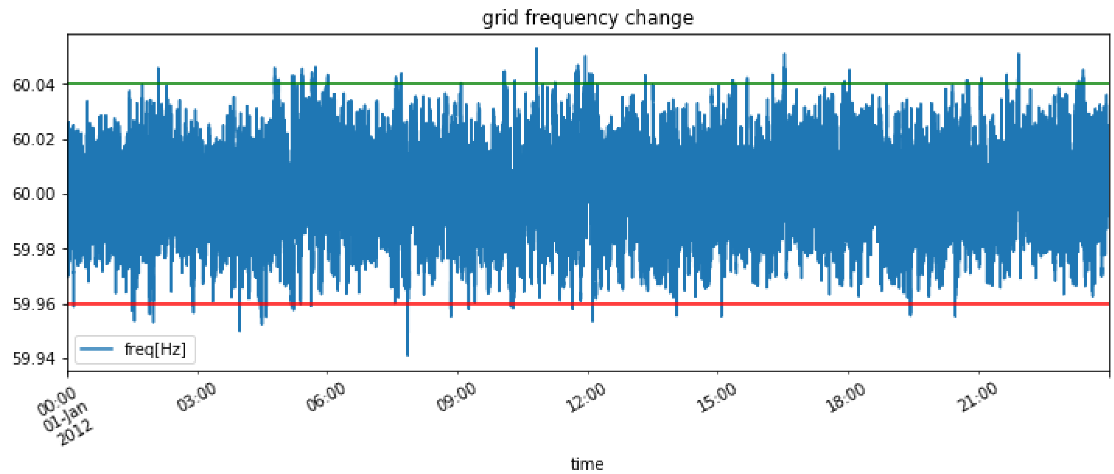
We can further plot the frequency data to get a better sense of it.

In [5]:
```
# install necessary packages
# uncomment the lines below if the system does not have them
# !pip3 install numpy
# !pip3 install pandas
# !pip3 install plotly
```

In [6]:
```
%matplotlib inline

import numpy as np
import pandas as pd

from plotly.offline import download_plotlyjs, init_notebook_mode,\
                           plot, iplot
import plotly.graph_objs as go
init_notebook_mode(connected=True)

raw_freq = pd.read_csv('frequency.PLAYER',index_col=0,\
                       names=['time','freq[Hz]'],
                       parse_dates=True,\
                       infer_datetime_format=True)

freq_low = 59.96
freq_high = 60.04
```

```
ax = raw_freq.plot(figsize=(12,4),rot=30,
                   title='grid frequency change')
ax.axhline(y=freq_low, c='red')
ax.axhline(y=freq_high, c='green')
```

Out[6]: <matplotlib.lines.Line2D at 0x10a3af240>



Next, we will run **local_gd/bin/gridlabd** on different **.glm** files and plot the outputs showing the difference with and without controllers.

We start with running **smSingle_base.glm**, which is almost same as the original **smSingle.glm** provided by NRECA to us with the main difference being that we changed the simulation clock and added a recoreder for waterheater1 at the end.

1 Base case

We begin with the same circuit provided by NRECA (smSingle.glm), and modify it slightly as follows:

- We change the simulation time to match the time of **frequency.PLAYER** and add a recorder to record the waterheater measurements and the ZIP load measurements (in this case, a fan). Note that we record data for waterheater1 as an example but it could be used for any waterheater.
- We also set the timestep to 1 second instead of 60 seconds.
- For a more realistic water draw schedule, we include a **hot_water_demand.glm** which exhibits typical the weekday and weekend water demand usage patterns.

Below we illustrate some of those changes made to the glm file:

In [7]:
```
# from 2012-01-01 to 2012-01-02
!head -9 smSingle_base.glm
```

```
clock {
        timezone PST+8PDT;
        starttime '2012-01-01 00:00:00';
        stoptime '2012-01-02 00:00:00';
};

#include "hot_water_demand.glm";

#set minimum_timestep=1;
```

In [8]:
```
# record data for waterheater1 and fan2(zipload) at 1s resolution
!tail -14 smSingle_base.glm
```

```
object recorder {
        interval 1;
        property base_power;
        file fan2_base.csv;
        parent fan2;
};

object recorder {
        interval 1;
        property measured_frequency,temperature,actual_load,is_waterheater_on,water_demand;
                // current_tank_status,waterheater_model,heatgain,power_state;
        file wh1_base.csv;
        parent waterheater1;
};
```

We are now ready to run a simulation with the base case (no control).

In [9]:
```
# run the gridlabd.bin to start the simulation
!local_gd/bin/gridlabd smSingle_base.glm
```

```
WARNING  [INIT] : waterheater::init() : height and diameter were not specified, defaulting to 3.

Core profiler results
======================

Total objects                 35 objects
Parallelism                    1 thread
Total time                  20.0 seconds
  Core time                  2.3 seconds (11.6%)
    Compiler                 1.2 seconds (5.9%)
    Instances                0.0 seconds (0.0%)
    Random variables         0.0 seconds (0.0%)
    Schedules                0.0 seconds (0.0%)
    Loadshapes               0.0 seconds (0.1%)
    Enduses                  0.0 seconds (0.1%)
```

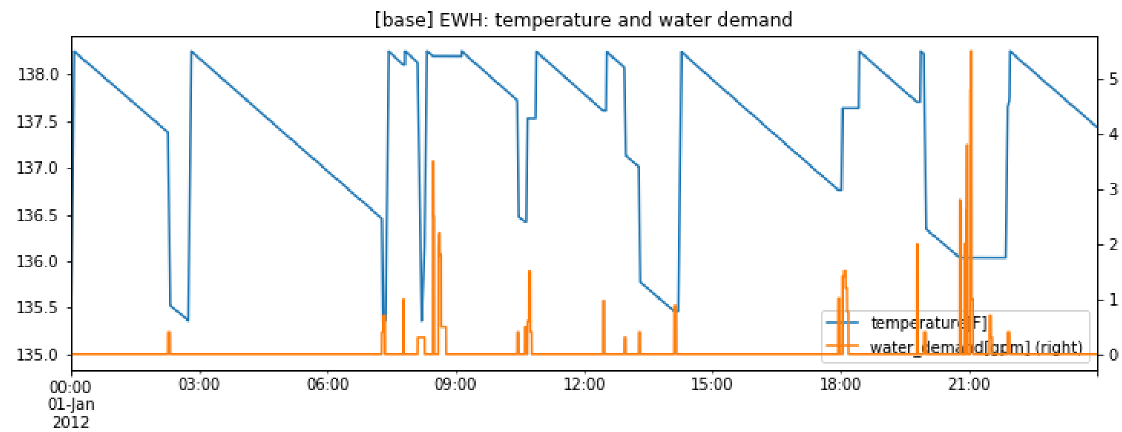
```
    Transforms                 0.2 seconds (1.0%)
  Model time                  17.7 seconds/thread (88.4%)
Simulation time                  1 days
Simulation speed                42 object.hours/second
Passes completed             86401 passes
Time steps completed         86401 timesteps
Convergence efficiency        1.00 passes/timestep
Read lock contention          0.0%
Write lock contention         0.0%
Average timestep                 1 seconds/timestep
Simulation rate               4320 x realtime


Model profiler results
======================

Class            Time (s) Time (%) msec/obj
---------------- -------- -------- --------
node               10.722    60.7%   5361.0
triplex_meter       1.095     6.2%    365.0
recorder            1.060     6.0%    353.3
house               0.893     5.1%    446.5
ZIPload             0.816     4.6%    102.0
waterheater         0.757     4.3%    378.5
transformer         0.675     3.8%    337.5
triplex_line        0.593     3.4%    296.5
regulator           0.388     2.2%    388.0
triplex_node        0.370     2.1%    370.0
auction             0.197     1.1%    197.0
climate             0.107     0.6%    107.0
================ ======== ======== ========
Total              17.673   100.0%    504.9

WARNING  [2012-01-02 00:00:00 PST] : last warning message was repeated 1 times
```

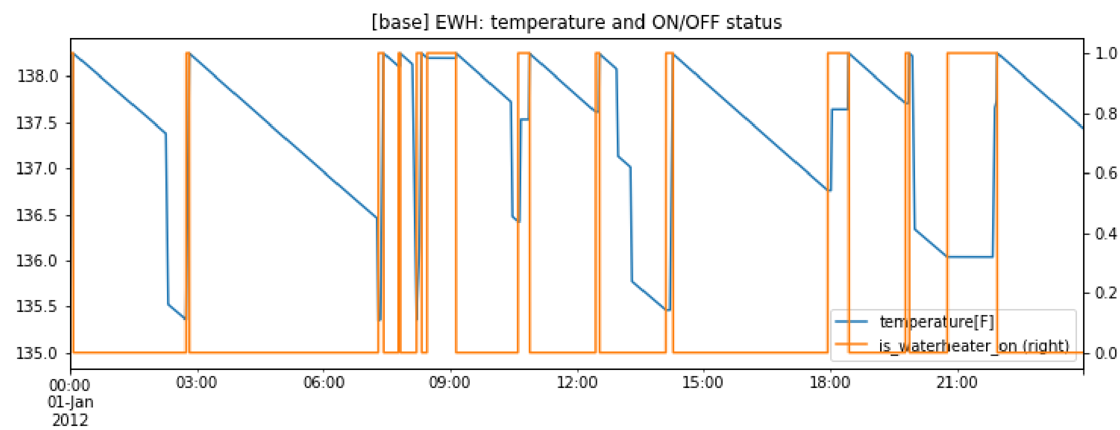
Now, we plot the generated waterheater data stored in **wh1_base.csv** and **fan2_base.csv** from the simulation.

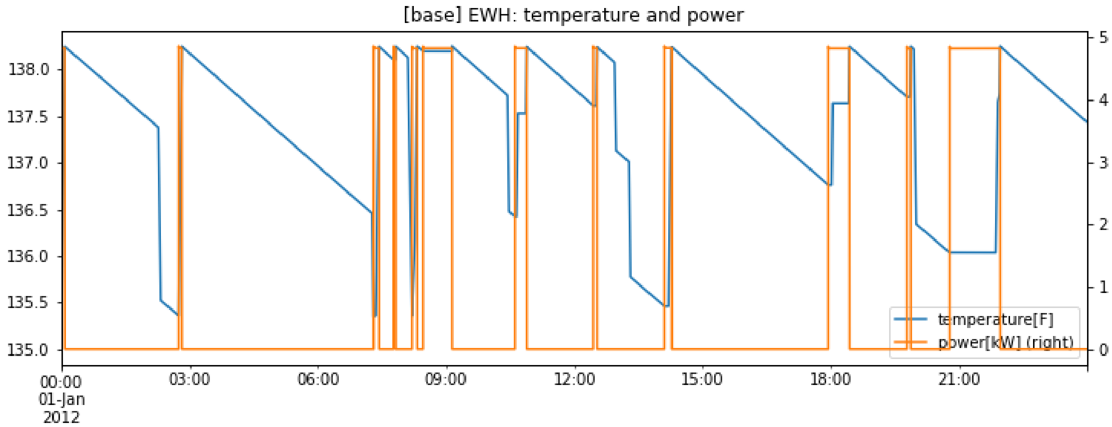
```
In [10]: df_base = pd.read_csv('wh1_base.csv',sep=',',header=8,
                     index_col=0,parse_dates=True,infer_datetime_format=True,
                     names=['freq[Hz]','temperature[F]','power[kW]',\
                         'is_waterheater_on','water_demand[gpm]'])

         df_base[['temperature[F]','water_demand[gpm]']].\
                 plot(figsize=(12,4),secondary_y='water_demand[gpm]',
                     title='[base] EWH: temperature and water demand')
```

```
df_base[['temperature[F]','is_waterheater_on']].\
        plot(figsize=(12,4),secondary_y='is_waterheater_on',
            title='[base] EWH: temperature and ON/OFF status')

df_base[['temperature[F]','power[kW]']].\
        plot(figsize=(12,4),secondary_y='power[kW]',
            title='[base] EWH: temperature and power')
```

Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x10e619f60>





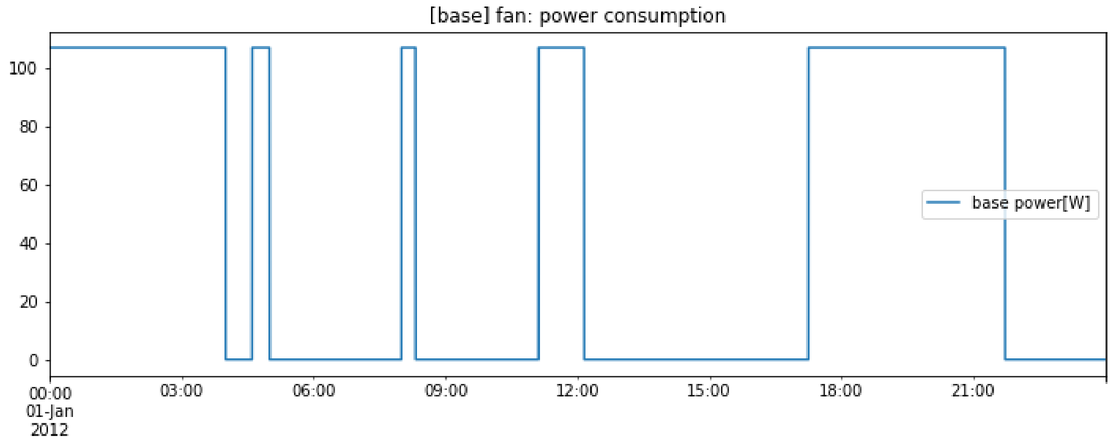



```
In [11]: # We can also plot the interactive version of the plot
         # during certain period
         def plotly_plotdf(df,title='Interactive plot of column variables'):
             if len(df)>20000:
                 print('Too many points, please reduce number of points!')
                 return
             data = []
             for i in df.columns:
                 trace = go.Scatter(
                     name = i,
                     x = df.index,
                     y = df[i]
                 )
                 data.append(trace)
             fig = go.Figure(
                 data = data,
                 layout = go.Layout(showlegend=True,
                                   title=title)
             )
             iplot(fig)
```

```
In [12]: # we can toggle the varialbe to visualize each of them
         # uncomment when you are running IPython notebook
         # plotly_plotdf(df_base.resample('1min').mean())
```

```
In [13]: df_base_fan = pd.read_csv('fan2_base.csv',sep=',',header=8,
                          index_col=0,parse_dates=True,
                          infer_datetime_format=True,
                          names=['base power[W]'])
         df_base_fan = df_base_fan*1000
         df_base_fan.plot(figsize=(12,4),
                         title='[base] fan: power consumption')
```

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x1067a9668>



2 Lenient Frequency Control

To configure the GridBallast controller, we set specific properties of the waterheater object in the glm file. The properties corresponding to the controller include:

- enable_freq_control [boolean]
- freq_lowlimit [float]
- freq_uplimit [float]
- enable_jitter [boolean]
- average_delay_time [integer]

For this test we modify waterheater 1 and fan 2 to enable the frequency control and set a wide frequency dead-band (59.9Hz - 60.1Hz). We expect the GridBallast controller to be rarely triggered.

In [14]: `!head -611 smSingle_lenient_freq.glm|tail -21`

```
object waterheater {
        schedule_skew -810;
        water_demand weekday_hotwater*1;
        name waterheater1;
        parent house1;
        heating_element_capacity 4.8 kW;
        thermostat_deadband 2.9;
        location INSIDE;
        tank_volume 50;
        tank_setpoint 136.8;
        tank_UA 2.4;
        temperature 135;
```

```
        object player {
                file frequency.PLAYER;
                property measured_frequency;
    };
        enable_freq_control true;
        freq_lowlimit 59.9;
        freq_uplimit 60.1;
        heat_mode ELECTRIC;
};
```

In [15]: !head -756 smSingle_lenient_freq.glm|tail -19

```
object ZIPload {
        name fan2;
        parent house2;
        power_fraction 0.013500;
        current_fraction 0.253400;
        base_power fan1*0.106899;
        impedance_pf 0.970000;
        current_pf 0.950000;
        power_pf -1.000000;
        impedance_fraction 0.733200;
        object player {
                file frequency.PLAYER;
                property measured_frequency;
        };
        enable_freq_control true;
        freq_lowlimit 59.9;
        freq_uplimit 60.1;
        groupid fan;
};
```

In [16]:
```
# run the gridlabd.bin to start the simulation
!local_gd/bin/gridlabd smSingle_lenient_freq.glm
```

```
WARNING  [INIT] : waterheater::init() : height and diameter were not specified, defaulting to 3.

Core profiler results
======================

Total objects                 37 objects
Parallelism                    1 thread
Total time                  23.0 seconds
  Core time                  2.9 seconds (12.8%)
    Compiler                 1.2 seconds (5.2%)
    Instances                0.0 seconds (0.0%)
    Random variables         0.0 seconds (0.0%)
```

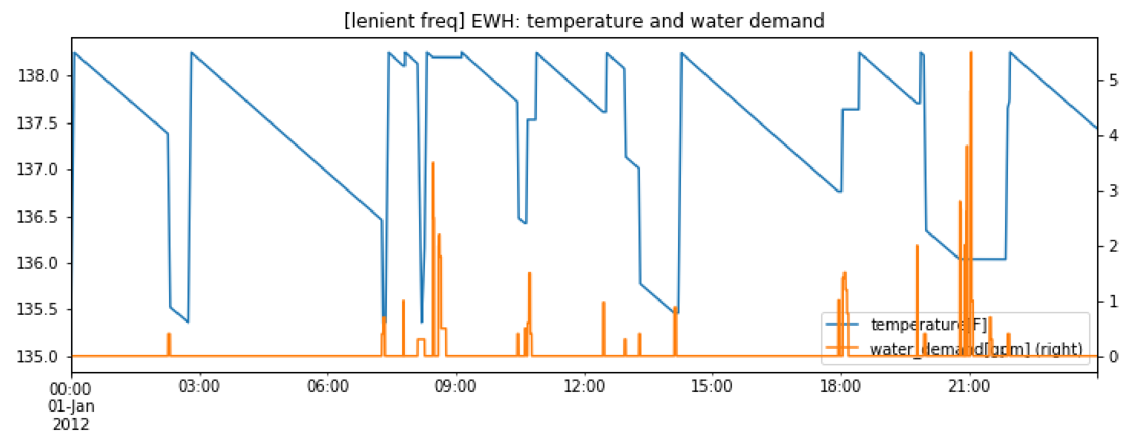
```
    Schedules                 0.0 seconds (0.0%)
    Loadshapes                0.0 seconds (0.1%)
    Enduses                   0.0 seconds (0.1%)
    Transforms                0.2 seconds (0.8%)
  Model time                 20.1 seconds/thread (87.2%)
Simulation time                 1 days
Simulation speed               39 object.hours/second
Passes completed            86401 passes
Time steps completed        86401 timesteps
Convergence efficiency       1.00 passes/timestep
Read lock contention         0.0%
Write lock contention        0.0%
Average timestep               1 seconds/timestep
Simulation rate             3757 x realtime


Model profiler results
======================

Class            Time (s) Time (%) msec/obj
---------------- -------- -------- --------
node               12.168    60.7%   6084.0
recorder            1.125     5.6%    375.0
triplex_meter       1.113     5.5%    371.0
house               0.906     4.5%    453.0
ZIPload             0.820     4.1%    102.5
player              0.784     3.9%    392.0
waterheater         0.747     3.7%    373.5
triplex_line        0.679     3.4%    339.5
transformer         0.660     3.3%    330.0
regulator           0.399     2.0%    399.0
triplex_node        0.330     1.6%    330.0
auction             0.216     1.1%    216.0
climate             0.110     0.5%    110.0
================ ======== ======== ========
Total              20.057   100.0%    542.1

WARNING  [2012-01-02 00:00:00 EST] : last warning message was repeated 1 times
```

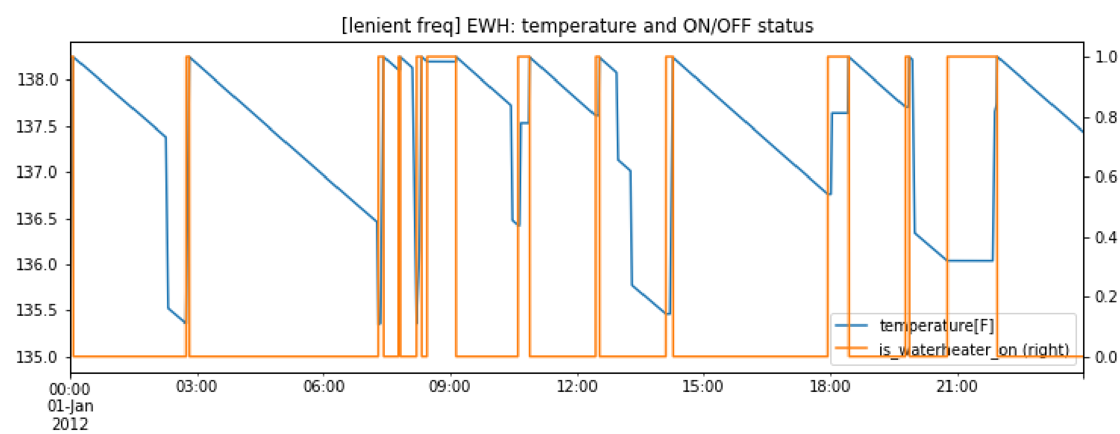
Now, we plot the generated waterheater data stored in **wh1_lenient_freq.csv** and **fan2_lenient_freq.csv** from the simulation.

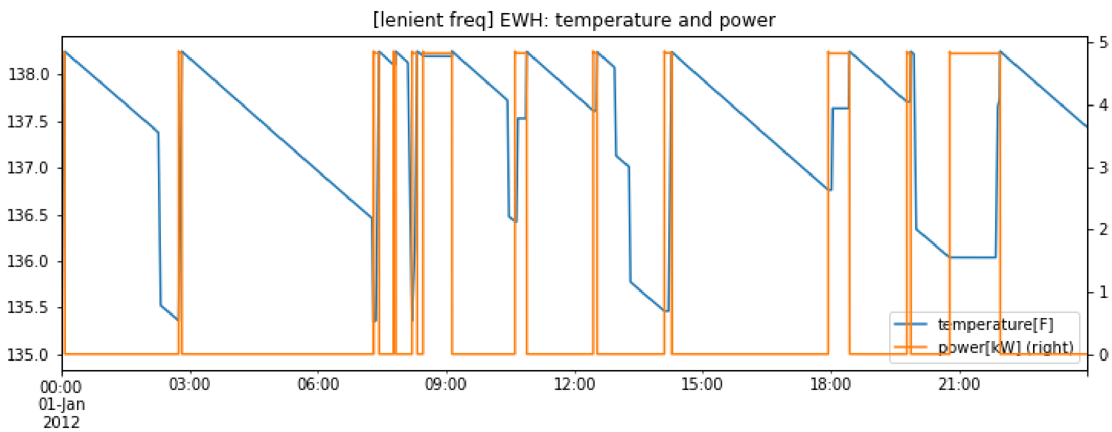
In [17]:
```python
# We save data to wh1_lenient_freq.csv and plot the results
df_lenient_freq = pd.read_csv('wh1_lenient_freq.csv',sep=',',
                          header=8,index_col=0,parse_dates=True,
                          infer_datetime_format=True,
                          names=['freq[Hz]','temperature[F]','power[kW]',
```

```
                            'is_waterheater_on','water_demand[gpm]'])

df_lenient_freq[['temperature[F]','water_demand[gpm]']].\
        plot(figsize=(12,4),secondary_y='water_demand[gpm]',
            title='[lenient freq] EWH: temperature and water demand')

df_lenient_freq[['temperature[F]','is_waterheater_on']].\
        plot(figsize=(12,4),secondary_y='is_waterheater_on',
            title='[lenient freq] EWH: temperature and ON/OFF status')

df_lenient_freq[['temperature[F]','power[kW]']].\
        plot(figsize=(12,4),secondary_y='power[kW]',
            title='[lenient freq] EWH: temperature and power')
```

Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x10f510c50>

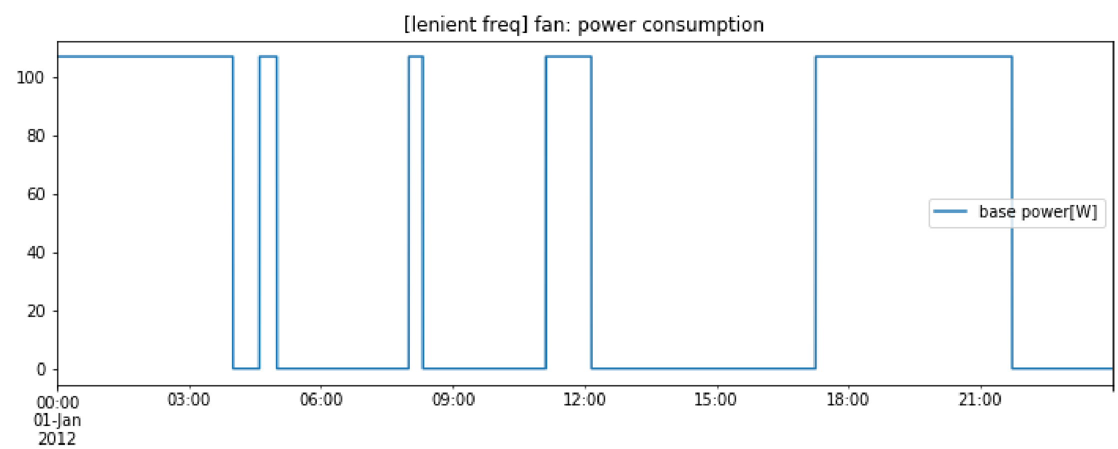







```
In [18]: df_lenient_fan = pd.read_csv('fan2_lenient_freq.csv',
                                      sep=',',header=8,
                                      index_col=0,parse_dates=True,
                                      infer_datetime_format=True,
                                      names=['base power[W]'])
         df_lenient_fan = df_lenient_fan*1000
         df_lenient_fan.plot(figsize=(12,4),
                title='[lenient freq] fan: power consumption')
```

```
Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x113944080>
```



3 Strict Frequency Control

We modify waterheater 1 and fan 2 to enable the frequency control, but we impose a tighter frequency deadband (59.97Hz - 60.03Hz). In other words, the gridballast controller should be triggered very often.

```
In [19]: !head -611 smSingle_strict_freq.glm|tail -21
```

```
object waterheater {
        schedule_skew -810;
        water_demand weekday_hotwater*1;
        name waterheater1;
        parent house1;
        heating_element_capacity 4.8 kW;
        thermostat_deadband 2.9;
        location INSIDE;
        tank_volume 50;
        tank_setpoint 136.8;
        tank_UA 2.4;
        temperature 135;
        object player {
                file frequency.PLAYER;
                property measured_frequency;
        };
        enable_freq_control true;
        freq_lowlimit 59.97;
        freq_uplimit 60.03;
        heat_mode ELECTRIC;
};
```

```
In [20]: !head -756 smSingle_strict_freq.glm|tail -19
```

```
object ZIPload {
        name fan2;
        parent house2;
        power_fraction 0.013500;
        current_fraction 0.253400;
        base_power fan1*0.106899;
        impedance_pf 0.970000;
        current_pf 0.950000;
        power_pf -1.000000;
        impedance_fraction 0.733200;
        object player {
                file frequency.PLAYER;
                property measured_frequency;
        };
        enable_freq_control true;
        freq_lowlimit 59.97;
        freq_uplimit 60.03;
        groupid fan;
};
```

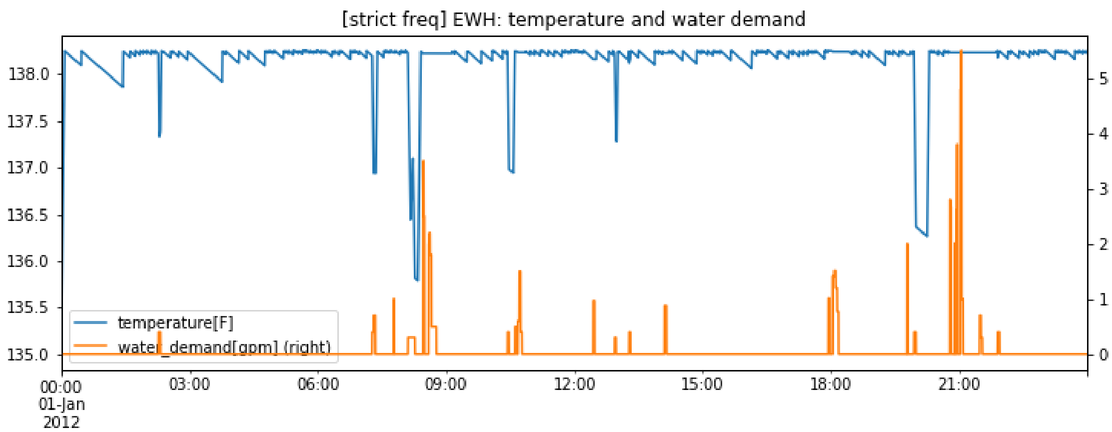
```
In [21]: # run the gridlabd.bin to start the simulation
         !local_gd/bin/gridlabd smSingle_strict_freq.glm
```

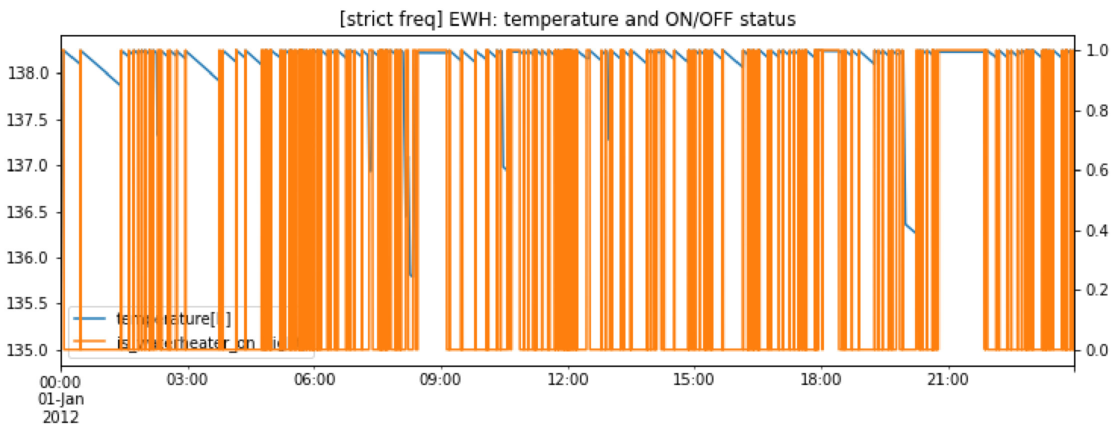
```
WARNING  [INIT] : waterheater::init() : height and diameter were not specified, defaulting to 3.

Core profiler results
======================

Total objects                 37 objects
Parallelism                    1 thread
Total time                  23.0 seconds
  Core time                  3.0 seconds (13.2%)
    Compiler                 1.2 seconds (5.1%)
    Instances                0.0 seconds (0.0%)
    Random variables         0.0 seconds (0.0%)
    Schedules                0.0 seconds (0.0%)
    Loadshapes               0.0 seconds (0.2%)
    Enduses                  0.0 seconds (0.1%)
    Transforms               0.2 seconds (0.9%)
  Model time                20.0 seconds/thread (86.8%)
Simulation time                1 days
Simulation speed              39 object.hours/second
Passes completed           86401 passes
Time steps completed       86401 timesteps
Convergence efficiency      1.00 passes/timestep
Read lock contention        0.0%
Write lock contention       0.0%
Average timestep               1 seconds/timestep
Simulation rate             3757 x realtime


Model profiler results
======================

Class            Time (s) Time (%) msec/obj
---------------- -------- -------- --------
node               11.945    59.8%   5972.5
recorder            1.159     5.8%    386.3
triplex_meter       1.138     5.7%    379.3
ZIPload             0.896     4.5%    112.0
house               0.891     4.5%    445.5
player              0.876     4.4%    438.0
waterheater         0.745     3.7%    372.5
transformer         0.641     3.2%    320.5
triplex_line        0.615     3.1%    307.5
regulator           0.437     2.2%    437.0
triplex_node        0.314     1.6%    314.0
auction             0.212     1.1%    212.0
climate             0.102     0.5%    102.0
================ ======== ======== ========
Total              19.971   100.0%    539.8
```

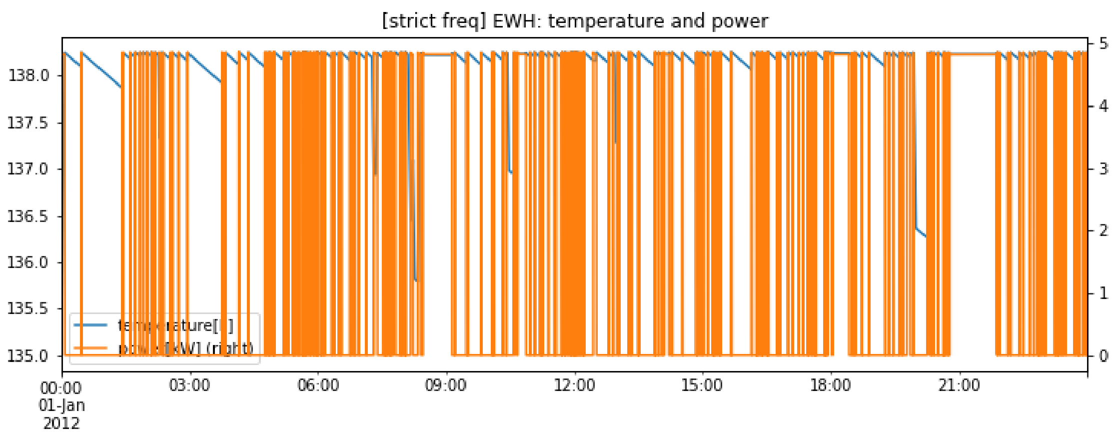
```
WARNING  [2012-01-02 00:00:00 EST] : last warning message was repeated 1 times
```

Now, we plot the generated waterheater data stored in **wh1_strict_freq.csv** and **fan2_strict_freq.csv** from the simulation.

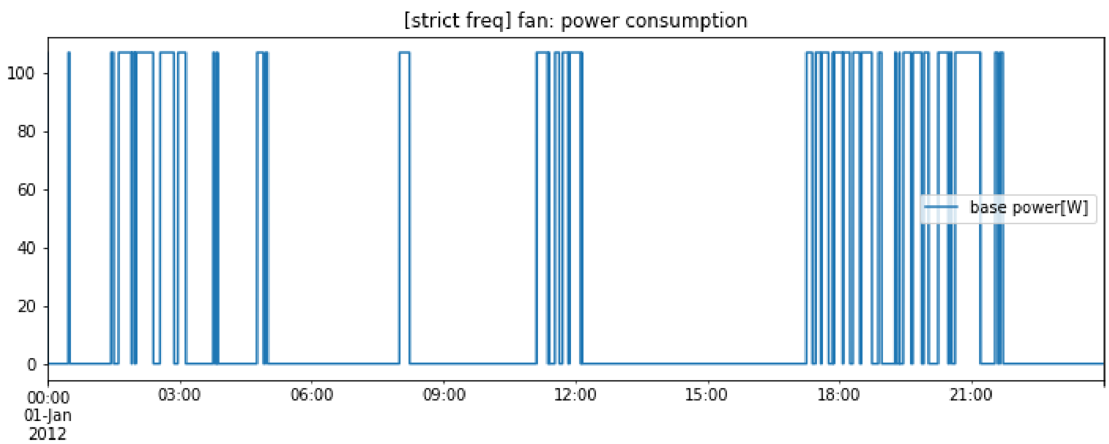
```
In [22]: # We save data to wh1_strict_freq.csv and plot the results
         df_strict_freq = pd.read_csv('wh1_strict_freq.csv',sep=',',
                                  header=8,index_col=0,parse_dates=True,
                                  infer_datetime_format=True,
                                  names=['freq[Hz]','temperature[F]','power[kW]',
                                         'is_waterheater_on','water_demand[gpm]'])

         df_strict_freq[['temperature[F]','water_demand[gpm]']].\
                 plot(figsize=(12,4),secondary_y='water_demand[gpm]',
                     title='[strict freq] EWH: temperature and water demand')

         df_strict_freq[['temperature[F]','is_waterheater_on']].\
                 plot(figsize=(12,4),secondary_y='is_waterheater_on',
                     title='[strict freq] EWH: temperature and ON/OFF status')

         df_strict_freq[['temperature[F]','power[kW]']].\
                 plot(figsize=(12,4),secondary_y='power[kW]',
                     title='[strict freq] EWH: temperature and power')

Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x10e6088d0>
```





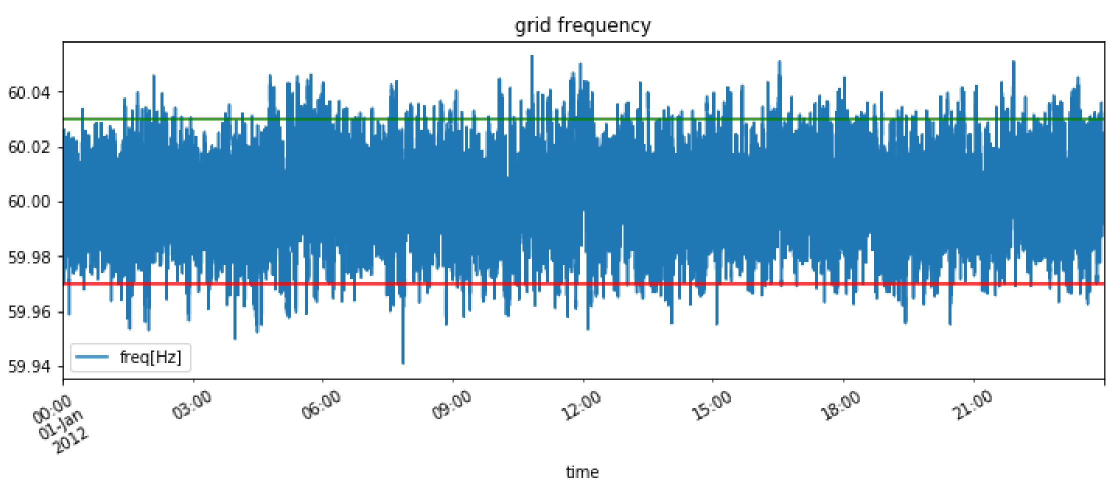



```
In [23]: df_strict_fan = pd.read_csv('fan2_strict_freq.csv',
                          sep=',',header=8,
                          index_col=0,parse_dates=True,
                          infer_datetime_format=True,
                          names=['base power[W]'])
         df_strict_fan = df_strict_fan*1000
         df_strict_fan.plot(figsize=(12,4),
                 title='[strict freq] fan: power consumption')

Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x10dac0748>
```




```
In [24]: # we can plot the frequency and the lower/upper limit again
         freq_low = 59.97
         freq_high = 60.03

         ax = raw_freq.plot(figsize=(12,4),rot=30,
                            title='grid frequency')
         ax.axhline(y=freq_low, c='red')
         ax.axhline(y=freq_high, c='green')
```

```
Out[24]: <matplotlib.lines.Line2D at 0x114d1db70>
```



4 Strict Frequency Control with Jitter (1 min)

We now modify the previous case (with a tight frequency deadband) and add a jitter to the response of the waterheater and fan, such that the start of GridBallast event will delay randomly with an expected value of 60 seconds (1 min). Internally, the controller delay follows a uniform distribution over the interval [1,2*average_delay_time].

We use 60 seconds to clearly illustrate the difference in the power consumption patterns of the water heater previously illustrated and this one with jitter control enabled. Needless to say, users can set these values differently depending on how many water heaters are connected to the network or other considerations.

```
In [25]: !head -613 smSingle_strict_freq_jitter60.glm|tail -23
```

```
object waterheater {
        schedule_skew -810;
        water_demand weekday_hotwater*1;
        name waterheater1;
        parent house1;
        heating_element_capacity 4.8 kW;
        thermostat_deadband 2.9;
        location INSIDE;
        tank_volume 50;
        tank_setpoint 136.8;
        tank_UA 2.4;
        temperature 135;
        object player {
                file frequency.PLAYER;
                property measured_frequency;
        };
        enable_freq_control true;
        freq_lowlimit 59.97;
        freq_uplimit 60.03;
        heat_mode ELECTRIC;
        enable_jitter true;
        average_delay_time 60;
};
```

```
In [26]: !head -760 smSingle_strict_freq_jitter60.glm|tail -21
```

```
object ZIPload {
        name fan2;
        parent house2;
        power_fraction 0.013500;
        current_fraction 0.253400;
        base_power fan1*0.106899;
        impedance_pf 0.970000;
        current_pf 0.950000;
        power_pf -1.000000;
        impedance_fraction 0.733200;
        object player {
                file frequency.PLAYER;
                property measured_frequency;
```

```
        };
        enable_freq_control true;
        freq_lowlimit 59.97;
        freq_uplimit 60.03;
        enable_jitter true;
        average_delay_time 60;
        groupid fan;
};
```

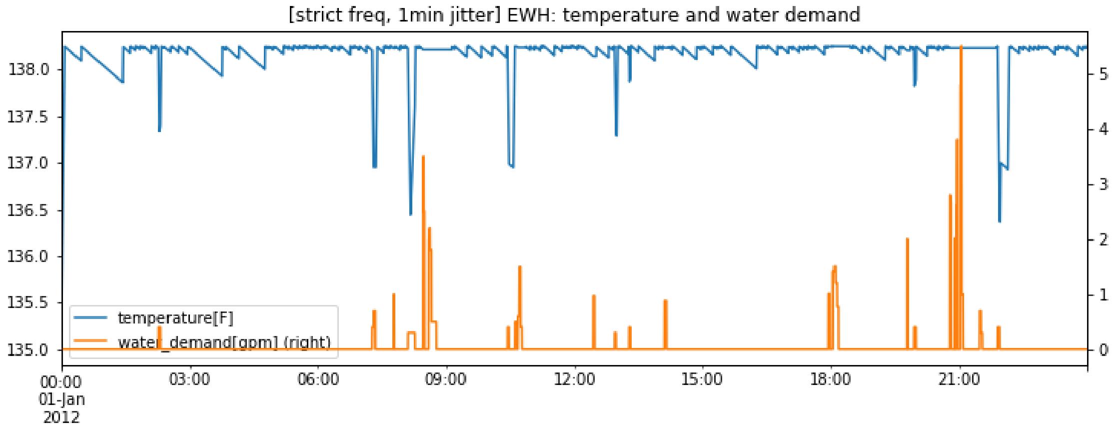
In [27]:
```
# run the gridlabd.bin to start the simulation
!local_gd/bin/gridlabd smSingle_strict_freq_jitter60.glm
```

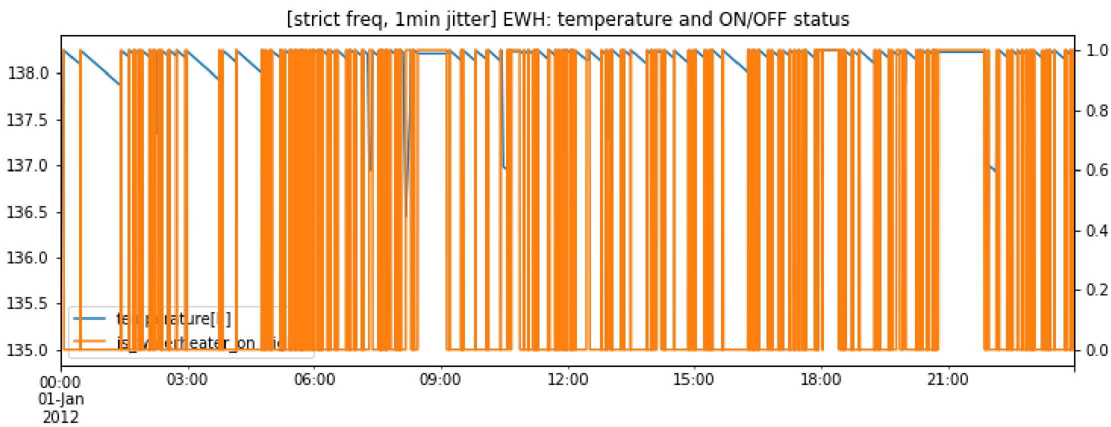
```
WARNING  [INIT] : waterheater::init() : height and diameter were not specified, defaulting to 3.

Core profiler results
======================

Total objects                 37 objects
Parallelism                    1 thread
Total time                  25.0 seconds
  Core time                  3.6 seconds (14.6%)
    Compiler                 1.1 seconds (4.5%)
    Instances                0.0 seconds (0.0%)
    Random variables         0.0 seconds (0.0%)
    Schedules                0.0 seconds (0.0%)
    Loadshapes               0.0 seconds (0.1%)
    Enduses                  0.0 seconds (0.1%)
    Transforms               0.2 seconds (0.8%)
  Model time                21.4 seconds/thread (85.4%)
Simulation time                1 days
Simulation speed              36 object.hours/second
Passes completed           86401 passes
Time steps completed       86401 timesteps
Convergence efficiency      1.00 passes/timestep
Read lock contention        0.0%
Write lock contention       0.0%
Average timestep               1 seconds/timestep
Simulation rate             3456 x realtime


Model profiler results
======================

Class            Time (s) Time (%) msec/obj
---------------- -------- -------- --------
node               13.104    61.4%   6552.0
recorder            1.211     5.7%    403.7
```

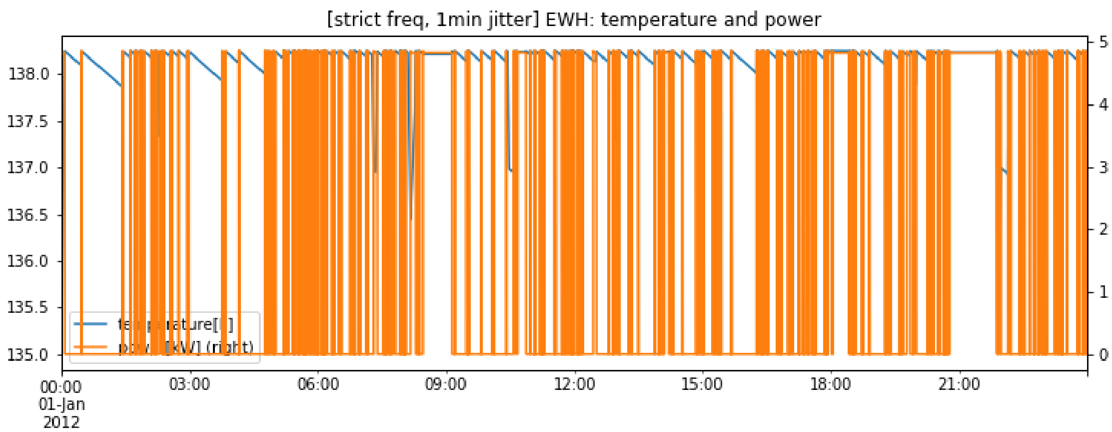
```
triplex_meter      1.131      5.3%    377.0
house              0.985      4.6%    492.5
player             0.866      4.1%    433.0
ZIPload            0.803      3.8%    100.4
waterheater        0.771      3.6%    385.5
triplex_line       0.686      3.2%    343.0
transformer        0.683      3.2%    341.5
regulator          0.456      2.1%    456.0
triplex_node       0.332      1.6%    332.0
auction            0.222      1.0%    222.0
climate            0.103      0.5%    103.0
================ ======== ======== ========
Total             21.353    100.0%    577.1

WARNING  [2012-01-02 00:00:00 EST] : last warning message was repeated 1 times
```

In [28]:
```
# We save data to wh1_strict_freq_jitter60.csv and plot the results
df_wh_jitter60 = pd.read_csv('wh1_strict_freq_jitter60.csv',sep=',',
                    header=8,index_col=0,parse_dates=True,
                    infer_datetime_format=True,
                    names=['freq[Hz]','temperature[F]','power[kW]',
                          'is_waterheater_on','water_demand[gpm]'])

df_wh_jitter60[['temperature[F]','water_demand[gpm]']].\
        plot(figsize=(12,4),secondary_y='water_demand[gpm]',
            title='[strict freq, 1min jitter] EWH: temperature and water demand')

df_wh_jitter60[['temperature[F]','is_waterheater_on']].\
        plot(figsize=(12,4),secondary_y='is_waterheater_on',
            title='[strict freq, 1min jitter] EWH: temperature and ON/OFF status')

df_wh_jitter60[['temperature[F]','power[kW]']].\
        plot(figsize=(12,4),secondary_y='power[kW]',
            title='[strict freq, 1min jitter] EWH: temperature and power')
```

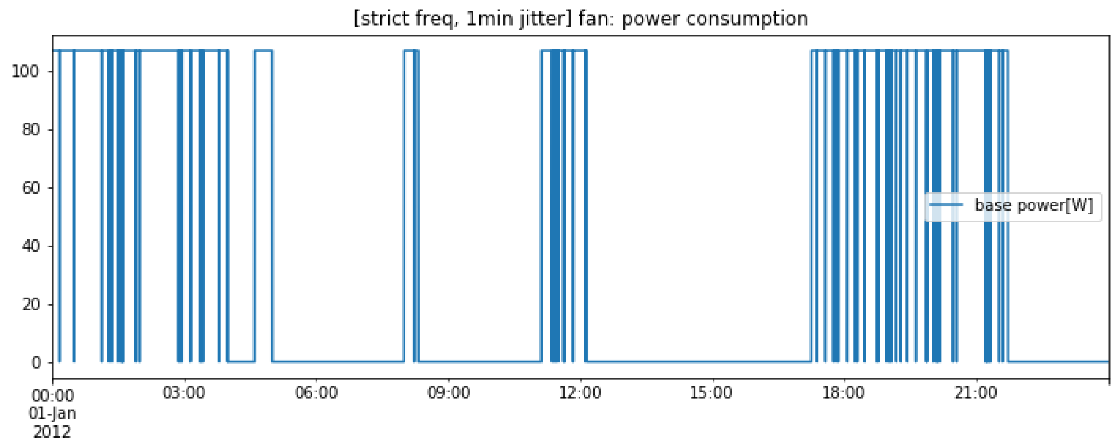
Out[28]: <matplotlib.axes._subplots.AxesSubplot at 0x117c94e48>


[strict freq, 1min jitter] EWH: temperature and water demand
temperature[F]
water_demand[gpm] (right)


[strict freq, 1min jitter] EWH: temperature and ON/OFF status

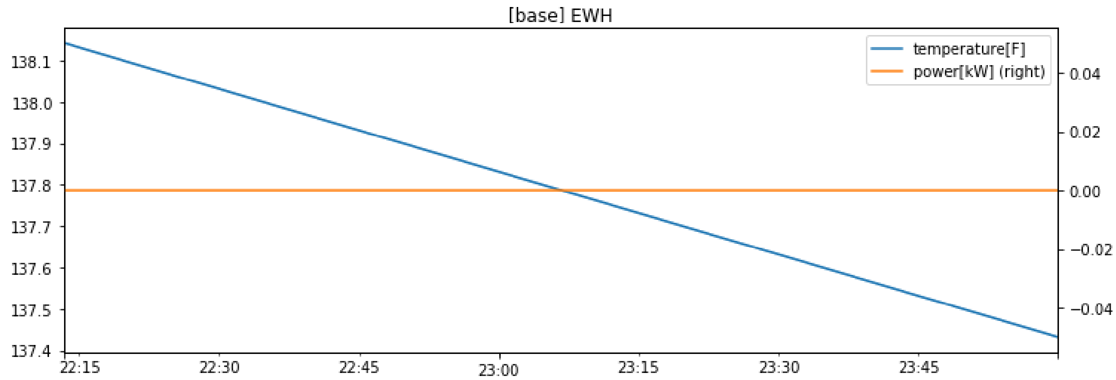

[strict freq, 1min jitter] EWH: temperature and power

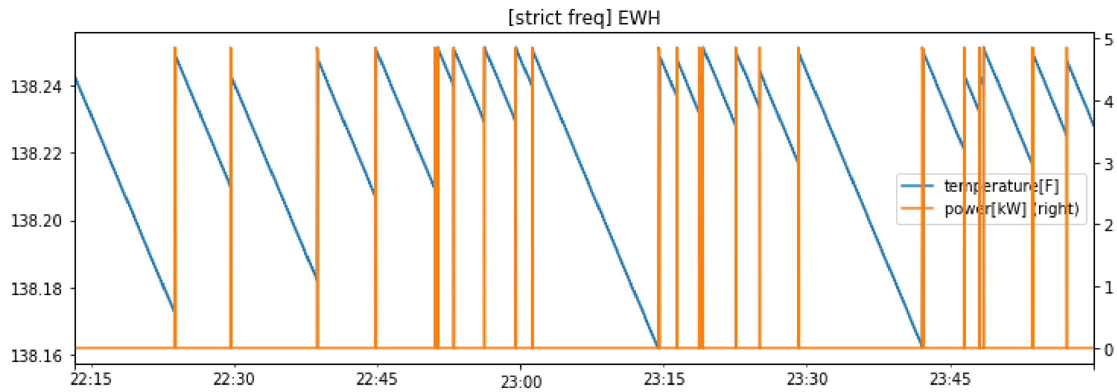
```
In [29]: df_fan_jitter60 = pd.read_csv('fan2_strict_freq_jitter60.csv',
                                       sep=',',
                                       header=8,index_col=0,parse_dates=True,
                                       infer_datetime_format=True,
                                       names=['base power[W]'])
         df_fan_jitter60 = df_fan_jitter60*1000
         df_fan_jitter60.plot(figsize=(12,4),
                 title='[strict freq, 1min jitter] fan: power consumption')

Out[29]: <matplotlib.axes._subplots.AxesSubplot at 0x118c85f98>
```

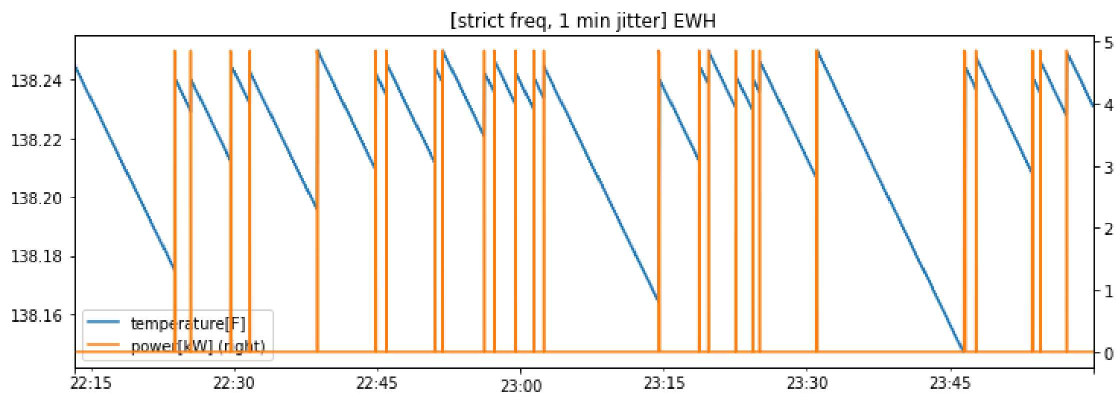


As we can see, after applying the jitter, the water heater should be engaged less often. However, since the jitter time is too short, we can barely see the difference unless we zoom in.

```
In [30]: # we look at jitter for water heater in shorter duration
         # As we can see, they behave slightly different
         df_base.iloc[80000:100000][['temperature[F]',
                 'power[kW]']].plot(figsize=(12,4),
                                    secondary_y='power[kW]',
                                   title='[base] EWH')
         df_strict_freq.iloc[80000:100000][['temperature[F]',
                 'power[kW]']].plot(figsize=(12,4),
                                    secondary_y='power[kW]',
                                   title='[strict freq] EWH')
         df_wh_jitter60.iloc[80000:100000][['temperature[F]',
                 'power[kW]']].plot(figsize=(12,4),
                                    secondary_y='power[kW]',
                         title='[strict freq, 1 min jitter] EWH')

Out[30]: <matplotlib.axes._subplots.AxesSubplot at 0x11ccfe940>
```



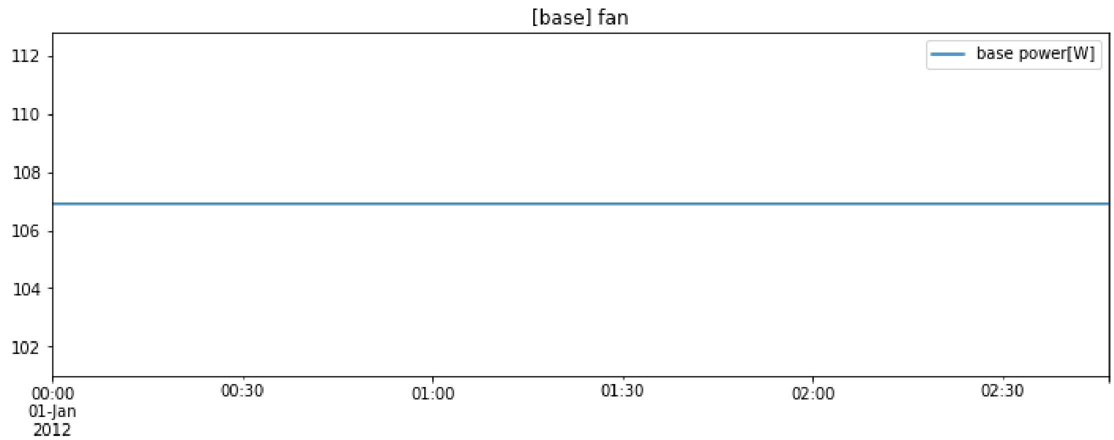


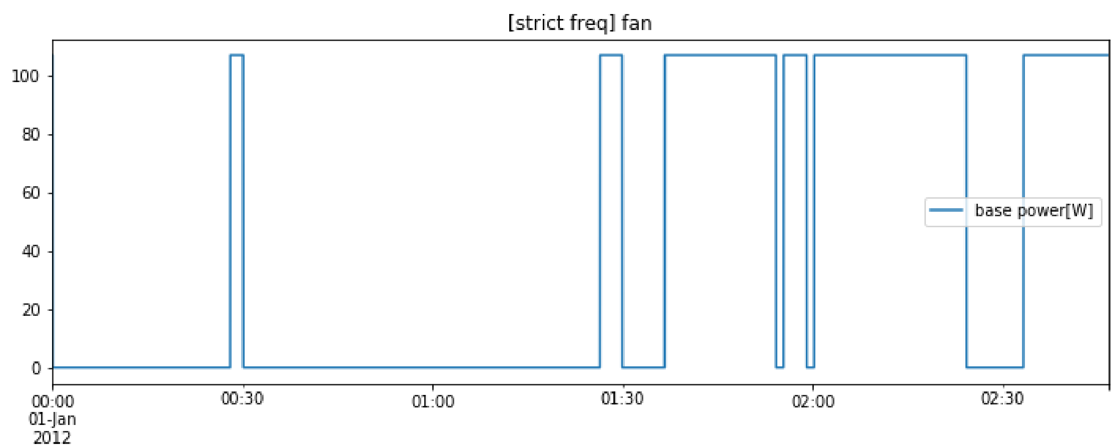


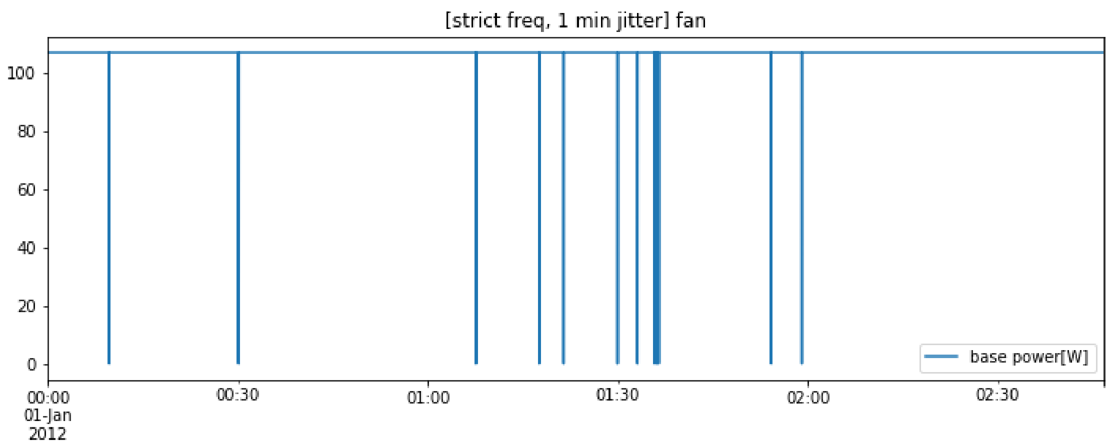

```
In [31]: # we look at jitter for the zipload in shorter duration as well
         # As we can see, they behave quiet differently since we don't need to consider
         # the thermal condition here. Once the frequency violation is detected, we can
```

```
# either turn on/turn off the load regardless of the origin schedule
df_base_fan.iloc[:10000].plot(figsize=(12,4),
                              title='[base] fan')
df_strict_fan.iloc[:10000].plot(figsize=(12,4),
                        title='[strict freq] fan')
df_fan_jitter60.iloc[:10000].plot(figsize=(12,4),
        title='[strict freq, 1 min jitter] fan')
```

Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0x11d4c4898>







As is seen, after applying the jitter, it tends to correct the power trace from strict frequency control case to the base case. It is obvious for the zipload[fan] case. Let's try the jitter with longer duration to see the same trend for the waterheater.

5 Strict Frequency Control with Jitter (10 mins)

We now modify the jitter such that the start of GridBallast event will delay randomly with an expected value of 600 seconds (10 mins) so that we can clearly see the jitter effects in the electric water heater as well.

```
In [32]: !head -613 smSingle_strict_freq_jitter600.glm|tail -23
```

```
object waterheater {
        schedule_skew -810;
        water_demand weekday_hotwater*1;
        name waterheater1;
        parent house1;
        heating_element_capacity 4.8 kW;
        thermostat_deadband 2.9;
        location INSIDE;
        tank_volume 50;
        tank_setpoint 136.8;
        tank_UA 2.4;
        temperature 135;
        object player {
                file frequency.PLAYER;
                property measured_frequency;
        };
        enable_freq_control true;
        freq_lowlimit 59.97;
        freq_uplimit 60.03;
        heat_mode ELECTRIC;
```

```
        enable_jitter true;
        average_delay_time 600;
};
```

In [33]: !head -760 smSingle_strict_freq_jitter600.glm|tail -21

```
object ZIPload {
        name fan2;
        parent house2;
        power_fraction 0.013500;
        current_fraction 0.253400;
        base_power fan1*0.106899;
        impedance_pf 0.970000;
        current_pf 0.950000;
        power_pf -1.000000;
        impedance_fraction 0.733200;
        object player {
                file frequency.PLAYER;
                property measured_frequency;
        };
        enable_freq_control true;
        freq_lowlimit 59.97;
        freq_uplimit 60.03;
        enable_jitter true;
        average_delay_time 600;
        groupid fan;
};
```

In [34]:
```
# run the gridlabd.bin to start the simulation
!local_gd/bin/gridlabd smSingle_strict_freq_jitter600.glm
```

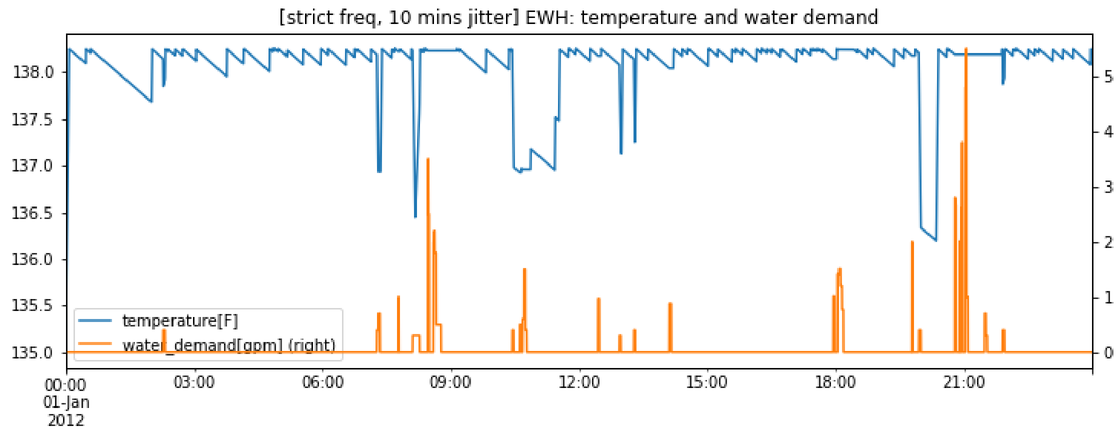
```
WARNING  [INIT] : waterheater::init() : height and diameter were not specified, defaulting to 3.

Core profiler results
======================

Total objects                 37 objects
Parallelism                    1 thread
Total time                  22.0 seconds
  Core time                  2.7 seconds (12.2%)
    Compiler                 1.3 seconds (5.9%)
    Instances                0.0 seconds (0.0%)
    Random variables         0.0 seconds (0.0%)
    Schedules                0.0 seconds (0.0%)
    Loadshapes               0.0 seconds (0.2%)
    Enduses                  0.0 seconds (0.1%)
    Transforms               0.2 seconds (0.8%)
```

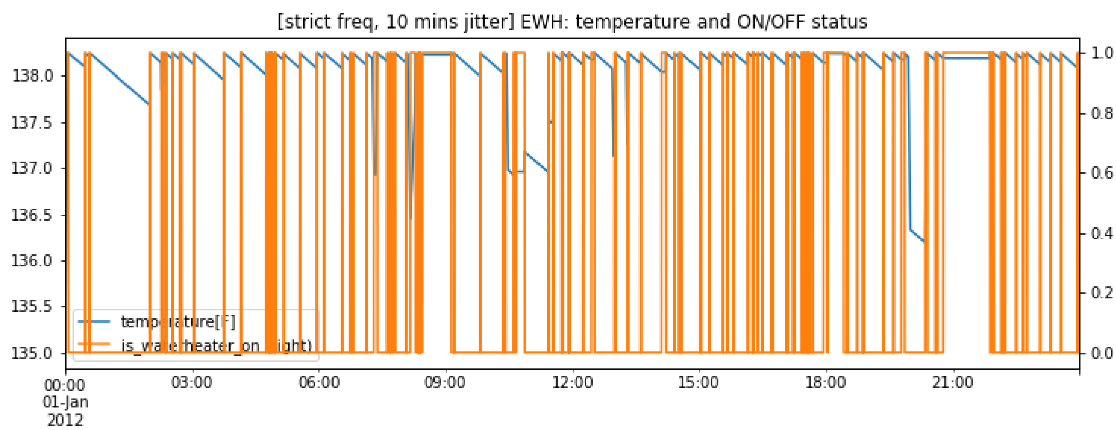
```
  Model time                  19.3 seconds/thread (87.8%)
Simulation time                  1 days
Simulation speed                40 object.hours/second
Passes completed             86401 passes
Time steps completed         86401 timesteps
Convergence efficiency        1.00 passes/timestep
Read lock contention          0.0%
Write lock contention         0.0%
Average timestep                 1 seconds/timestep
Simulation rate              3927 x realtime


Model profiler results
======================

Class            Time (s) Time (%) msec/obj
---------------- -------- -------- --------
node               11.362    58.8%   5681.0
triplex_meter       1.138     5.9%    379.3
recorder            1.095     5.7%    365.0
ZIPload             0.889     4.6%    111.1
house               0.862     4.5%    431.0
player              0.849     4.4%    424.5
waterheater         0.705     3.7%    352.5
transformer         0.678     3.5%    339.0
triplex_line        0.664     3.4%    332.0
regulator           0.406     2.1%    406.0
triplex_node        0.334     1.7%    334.0
auction             0.214     1.1%    214.0
climate             0.117     0.6%    117.0
================ ======== ======== ========
Total              19.313   100.0%    522.0

WARNING  [2012-01-02 00:00:00 EST] : last warning message was repeated 1 times
```

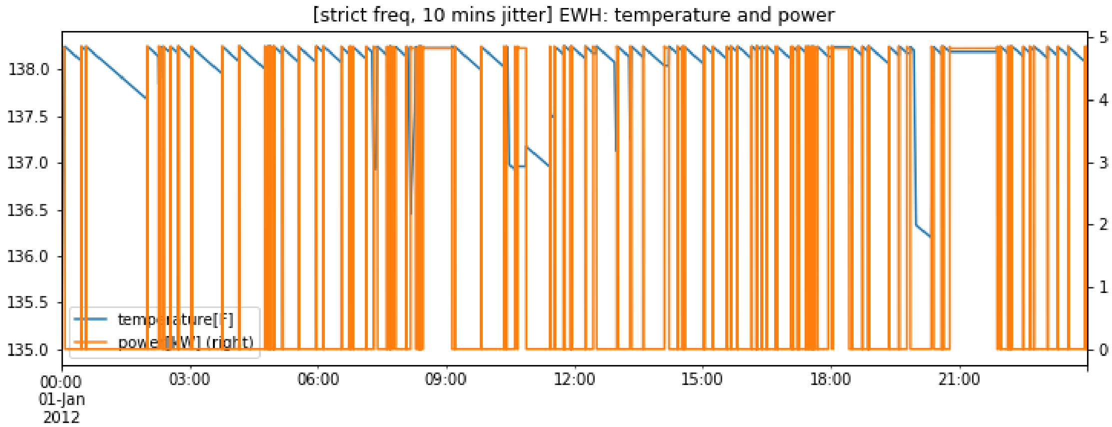
In [35]:
```python
# We save data to wh1_strict_freq_jitter600.csv and plot the results
df_wh_jitter600 = pd.read_csv('wh1_strict_freq_jitter600.csv',sep=',',
                  header=8,index_col=0,parse_dates=True,
                  infer_datetime_format=True,
                  names=['freq[Hz]','temperature[F]','power[kW]',
                        'is_waterheater_on','water_demand[gpm]'])

df_wh_jitter600[['temperature[F]','water_demand[gpm]']].\
        plot(figsize=(12,4),secondary_y='water_demand[gpm]',
            title='[strict freq, 10 mins jitter] EWH: temperature and water demand')

df_wh_jitter600[['temperature[F]','is_waterheater_on']].\
```

```
            plot(figsize=(12,4),secondary_y='is_waterheater_on',
                title='[strict freq, 10 mins jitter] EWH: temperature and ON/OFF status')

    df_wh_jitter600[['temperature[F]','power[kW]']].\
            plot(figsize=(12,4),secondary_y='power[kW]',
                title='[strict freq, 10 mins jitter] EWH: temperature and power')
```

Out[35]: <matplotlib.axes._subplots.AxesSubplot at 0x11cf73cf8>

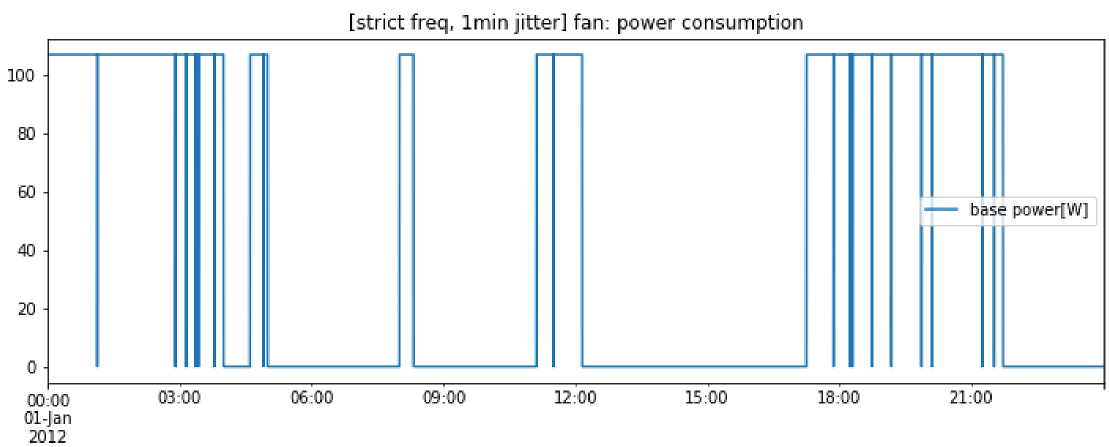







```
In [36]: df_fan_jitter600 = pd.read_csv('fan2_strict_freq_jitter600.csv',
                                        sep=',',
                                        header=8,index_col=0,parse_dates=True,
                                        infer_datetime_format=True,
                                        names=['base power[W]'])
         df_fan_jitter600 = df_fan_jitter600*1000
         df_fan_jitter600.plot(figsize=(12,4),
                 title='[strict freq, 1min jitter] fan: power consumption')
```

```
Out[36]: <matplotlib.axes._subplots.AxesSubplot at 0x10e98a3c8>
```

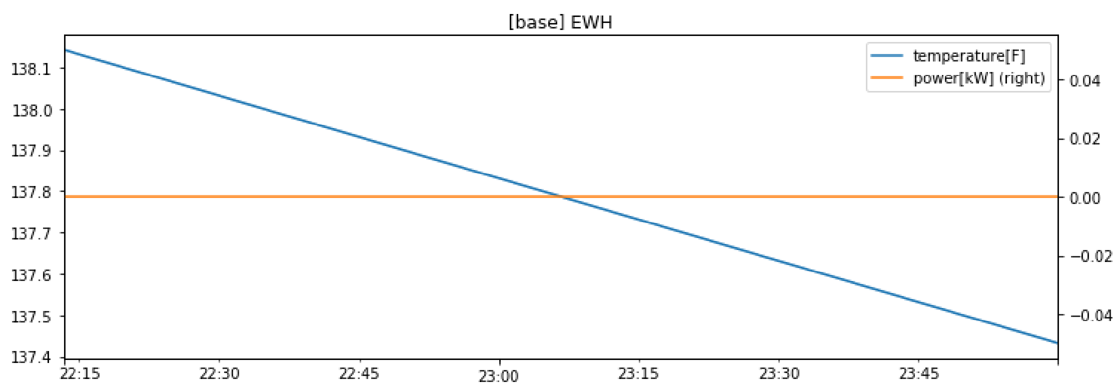


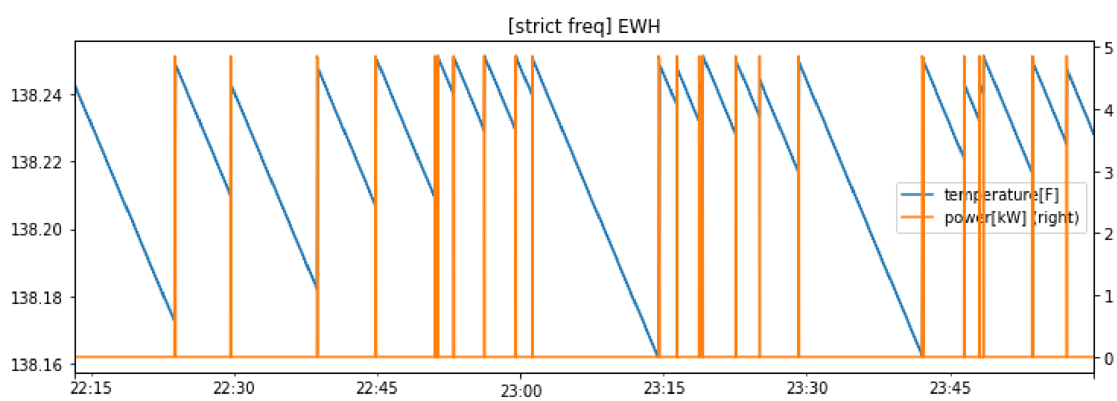
As we can see, after applying the 10 min jitter, now the water heater is engaged less often than in the previous experiment without jitter.

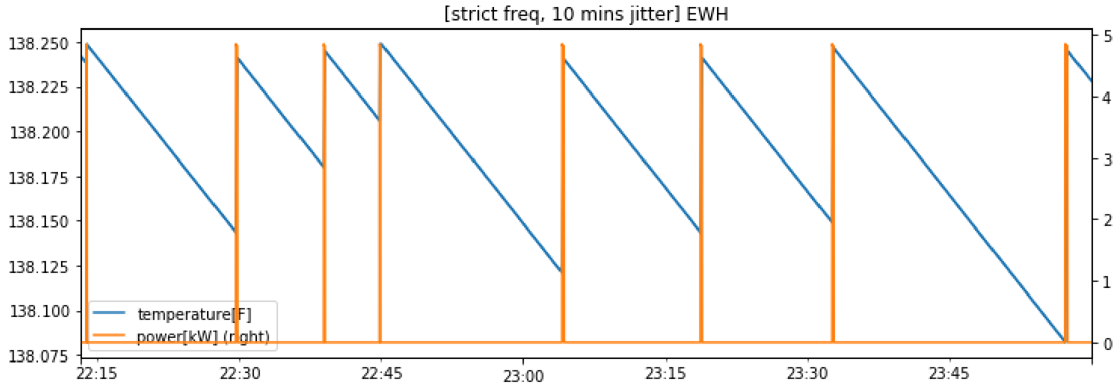
As we did in previous examples, we now look into a shorter duration to better understand the effect of the jitter.

In [37]:
```
# we look at jitter for the electric water heater in shorter duration
# As we can see, they behave slightly different, the one with jitter behaves
# more like the one without frequency control (base case)
df_base.iloc[80000:100000][['temperature[F]',
        'power[kW]']].plot(figsize=(12,4),
                           secondary_y='power[kW]',
                          title='[base] EWH')
df_strict_freq.iloc[80000:100000][['temperature[F]',
        'power[kW]']].plot(figsize=(12,4),
                           secondary_y='power[kW]',
                          title='[strict freq] EWH')
df_wh_jitter600.iloc[80000:100000][['temperature[F]',
        'power[kW]']].plot(figsize=(12,4),
                           secondary_y='power[kW]',
            title='[strict freq, 10 mins jitter] EWH')
```

Out[37]: <matplotlib.axes._subplots.AxesSubplot at 0x11c0c0780>

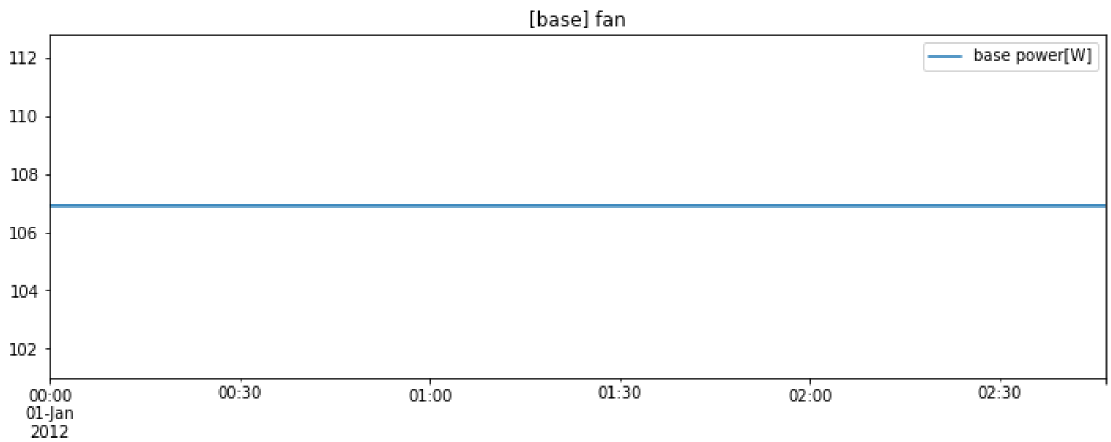


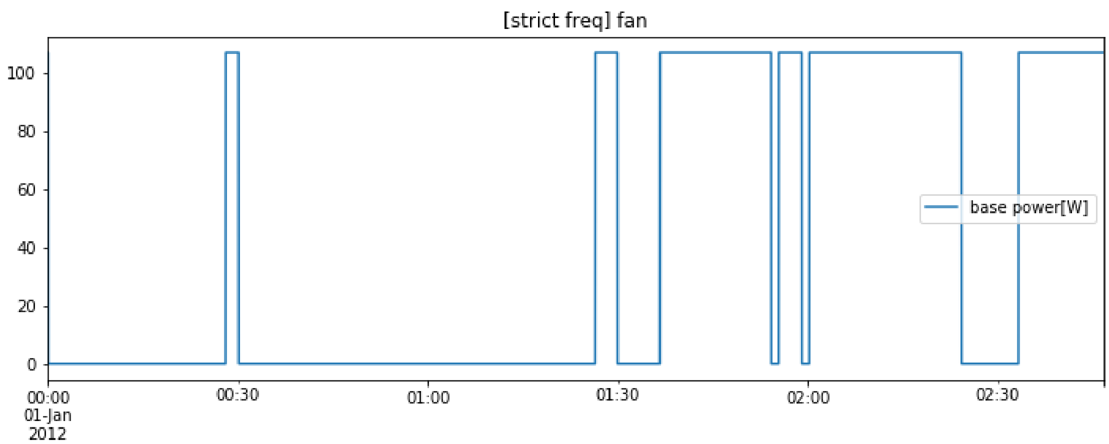


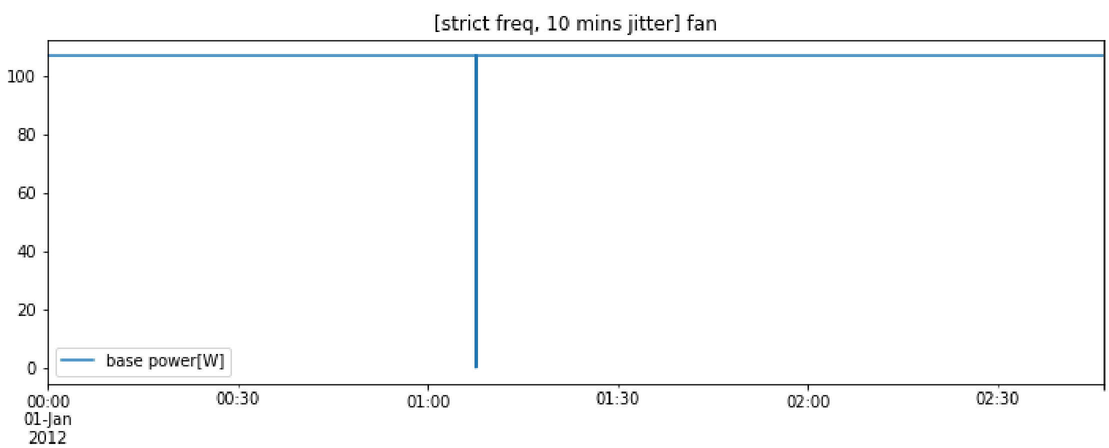



```
In [38]: # we look at jitter for the zipload in shorter duration as well
         # As we can see, when we apply jitter, it behaves more like
         # the one without frequency control, and the longer the jitter duration,
         # the more likely the power trace becomes to the one with out frequency control
         df_base_fan.iloc[:10000].plot(figsize=(12,4),
                                       title='[base] fan')
         df_strict_fan.iloc[:10000].plot(figsize=(12,4),
                                  title='[strict freq] fan')
         df_fan_jitter600.iloc[:10000].plot(figsize=(12,4),
                 title='[strict freq, 10 mins jitter] fan')
```

```
Out[38]: <matplotlib.axes._subplots.AxesSubplot at 0x10c0abd30>
```







6 Lenient Frequency Control & Lock Mode Enabled

We will use a very simple example to demonstrate how to enable the lock for ON/OFF during certain period. For example, if we want to enable lock between 18:00-22:00, and force load ON between 19:00-21:00, and force load OFF between from 18:00-19:00 and 21:00-22:00, we can specify a schedule file like this.

```
In [39]: cat lock_mode_schedule.glm
```

```
schedule temp_lock_enable {
        * 0-17 * * * 0;
        * 18-21 * * * 1;
        * 22-23 * * * 0;
};
```

```
schedule temp_lock_status {
        * 18 * * * 0;
        * 19-20 * * * 1;
        * 21 * * * 0;
};
```

In [40]:
```
# we decide not to override the thermostat setpoint by letting lock_OVERRIDE_TS to be f
# we can let this variable to be true if we want a very strict control of the TCLs
!head -616 smSingle_lenient_freq_lock_mode.glm|tail -26
```

```
object waterheater {
        schedule_skew -810;
        water_demand weekday_hotwater*1;
        name waterheater1;
        parent house1;
        heating_element_capacity 4.8 kW;
        thermostat_deadband 2.9;
        location INSIDE;
        tank_volume 50;
        tank_setpoint 136.8;
        tank_UA 2.4;
        temperature 135;
        object player {
                file frequency.PLAYER;
                property measured_frequency;
    };
        enable_freq_control true;
        freq_lowlimit 59.9;
        freq_uplimit 60.1;
        heat_mode ELECTRIC;
        enable_lock_mode temp_lock_enable;
        lock_OVERRIDE_TS false;
        lock_STATUS temp_lock_status;
};
```

In [41]:
```
!head -763 smSingle_lenient_freq_lock_mode.glm|tail -22
```

```
object ZIPload {
        name fan2;
        parent house2;
        power_fraction 0.013500;
        current_fraction 0.253400;
        base_power fan1*0.106899;
        impedance_pf 0.970000;
        current_pf 0.950000;
```

```
        power_pf -1.000000;
        impedance_fraction 0.733200;
        object player {
                file frequency.PLAYER;
                property measured_frequency;
        };
        enable_freq_control true;
        freq_lowlimit 59.9;
        freq_uplimit 60.1;
        enable_lock_mode temp_lock_enable;
        lock_STATUS temp_lock_status;
        groupid fan;
};
```

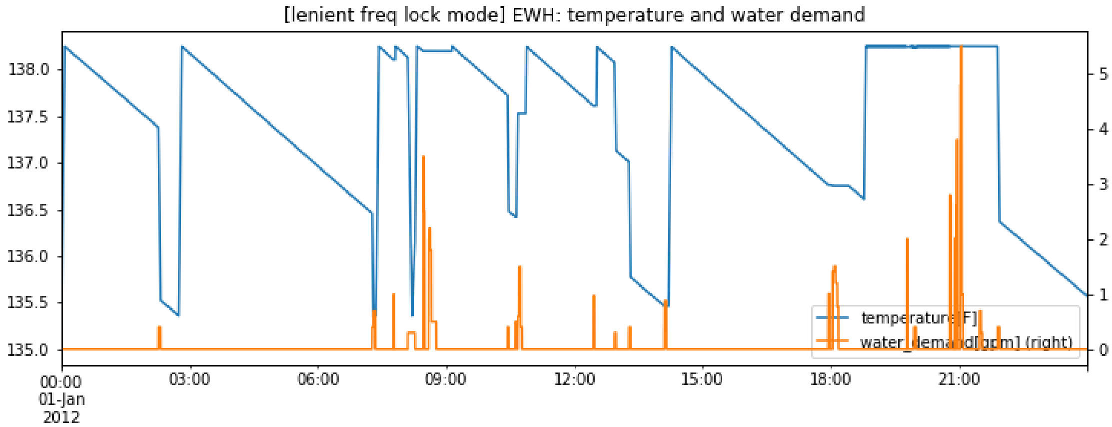
In [42]:
```
# run the gridlabd.bin to start the simulation
!local_gd/bin/gridlabd smSingle_lenient_freq_lock_mode.glm
```

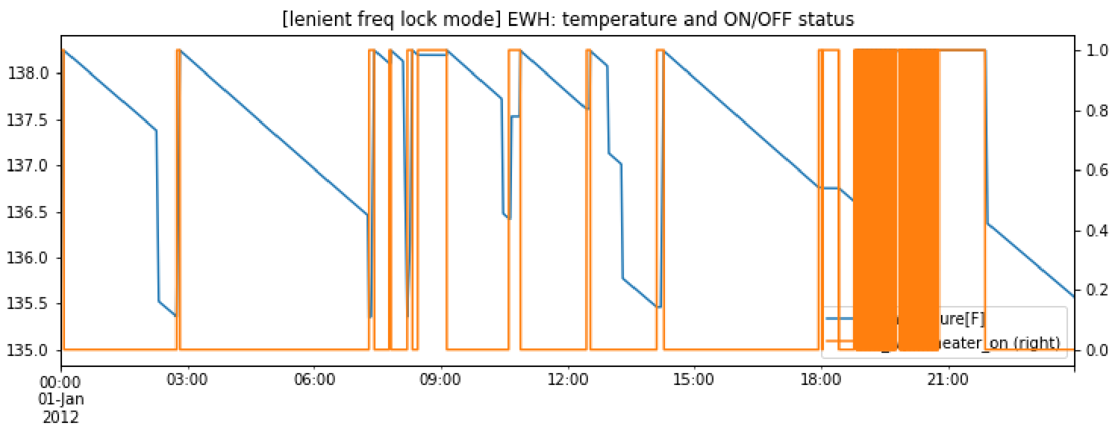
```
WARNING  [INIT] : waterheater::init() : height and diameter were not specified, defaulting to 3.

Core profiler results
======================

Total objects                 37 objects
Parallelism                    1 thread
Total time                  23.0 seconds
  Core time                  2.2 seconds (9.4%)
    Compiler                 1.4 seconds (6.1%)
    Instances                0.0 seconds (0.0%)
    Random variables         0.0 seconds (0.0%)
    Schedules                0.0 seconds (0.0%)
    Loadshapes               0.0 seconds (0.1%)
    Enduses                  0.0 seconds (0.2%)
    Transforms               0.3 seconds (1.1%)
  Model time                20.8 seconds/thread (90.6%)
Simulation time                1 days
Simulation speed              39 object.hours/second
Passes completed           86401 passes
Time steps completed       86401 timesteps
Convergence efficiency      1.00 passes/timestep
Read lock contention        0.0%
Write lock contention       0.0%
Average timestep               1 seconds/timestep
Simulation rate             3757 x realtime


Model profiler results
```

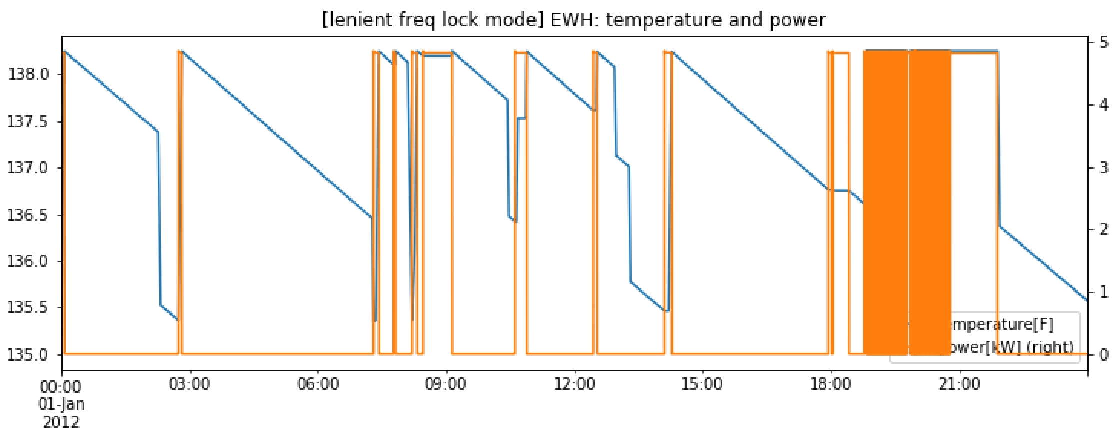
```
======================

Class             Time (s) Time (%) msec/obj
---------------- -------- -------- --------
node               12.364    59.4%   6182.0
recorder            1.237     5.9%    412.3
triplex_meter       1.203     5.8%    401.0
house               0.962     4.6%    481.0
player              0.933     4.5%    466.5
ZIPload             0.881     4.2%    110.1
waterheater         0.793     3.8%    396.5
transformer         0.676     3.2%    338.0
triplex_line        0.663     3.2%    331.5
regulator           0.424     2.0%    424.0
triplex_node        0.340     1.6%    340.0
auction             0.236     1.1%    236.0
climate             0.118     0.6%    118.0
================ ======== ======== ========
Total              20.830   100.0%    563.0

WARNING  [2012-01-02 00:00:00 EST] : last warning message was repeated 1 times
```

Now, we plot the generated waterheater data stored in **wh1_lenient_freq_lock_mode.csv** and **fan2_lenient_freq_lock_mode.csv** from the simulation.

```
In [43]: # We save data to wh1_lenient_freq.csv and plot the results
         df_lenient_freq_lk = pd.read_csv('wh1_lenient_freq_lock_mode.csv',sep=',',
                           header=8,index_col=0,parse_dates=True,
                           infer_datetime_format=True,
                           names=['freq[Hz]','temperature[F]','power[kW]',
                                 'is_waterheater_on','water_demand[gpm]'])

         df_lenient_freq_lk[['temperature[F]','water_demand[gpm]']].\
                 plot(figsize=(12,4),secondary_y='water_demand[gpm]',
                     title='[lenient freq lock mode] EWH: temperature and water demand')

         df_lenient_freq_lk[['temperature[F]','is_waterheater_on']].\
                 plot(figsize=(12,4),secondary_y='is_waterheater_on',
                     title='[lenient freq lock mode] EWH: temperature and ON/OFF status')

         df_lenient_freq_lk[['temperature[F]','power[kW]']].\
                 plot(figsize=(12,4),secondary_y='power[kW]',
                     title='[lenient freq lock mode] EWH: temperature and power')

Out[43]: <matplotlib.axes._subplots.AxesSubplot at 0x120157b70>
```


[lenient freq lock mode] EWH: temperature and water demand
138.0
137.5
137.0
136.5
136.0
135.5
135.0
5
4
3
2
1
0
00:00
01-Jan
2012
03:00
06:00
09:00
12:00
15:00
18:00
21:00


[lenient freq lock mode] EWH: temperature and ON/OFF status
138.0
137.5
137.0
136.5
136.0
135.5
135.0
1.0
0.8
0.6
0.4
0.2
0.0
00:00
01-Jan
2012
03:00
06:00
09:00
12:00
15:00
18:00
21:00


[lenient freq lock mode] EWH: temperature and power
138.0
137.5
137.0
136.5
136.0
135.5
135.0
5
4
3
2
1
0
00:00
01-Jan
2012
03:00
06:00
09:00
12:00
15:00
18:00
21:00

As we can see, the load is off starting from 18:00, however, due to the water usage events, the temperature set point has a higher priority, and the load is ON to maintain the temperature within the dead band.

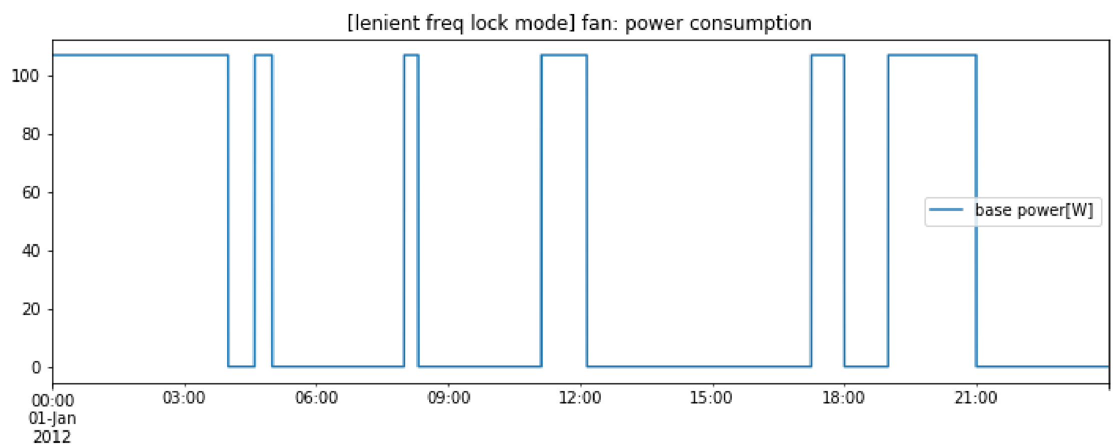
Starting from 19:00, the load is forced ON, however, once the temperature reaches the upper band, the load is forced OFF, that is why we see the dense fluctuations between 19:00-21:00.

Starting from 21:00, the load is supposed to be OFF, however, due to the temperature setting point has a higher priority, the load is forced to be ON to maintain the proper temperature.

Now let's look at the fan.

```
In [44]: df_lenient_fan_lk = pd.read_csv('fan2_lenient_freq_lock_mode.csv',
                                 sep=',',header=8,
                                 index_col=0,parse_dates=True,
                                 infer_datetime_format=True,
                                 names=['base power[W]'])
         df_lenient_fan_lk = df_lenient_fan_lk*1000
         df_lenient_fan_lk.plot(figsize=(12,4),
                 title='[lenient freq lock mode] fan: power consumption')
```

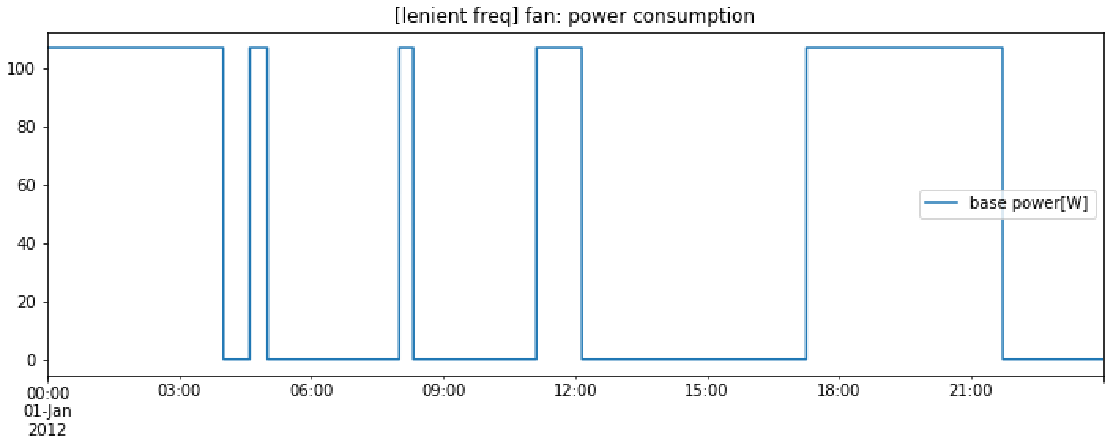
```
Out[44]: <matplotlib.axes._subplots.AxesSubplot at 0x11d3cf4e0>
```



The fan is quite properly behaved, it is OFF between 18:00-19:00, ON between 19:00-21:00, OFF again between 21:00-22:00. Exactly as the lock mode schedule specified. We can also look at the origin power trace for comparison.

```
In [45]: df_lenient_fan.plot(figsize=(12,4),
                 title='[lenient freq] fan: power consumption')
```

```
Out[45]: <matplotlib.axes._subplots.AxesSubplot at 0x112421358>
```


[lenient freq] fan: power consumption
100
80
60
40
20
0
00:00
01-Jan
2012
03:00
06:00
09:00
12:00
15:00
18:00
21:00
base power[W]