Energy Consumption Disaggregation Using Appliance Profiles

Nathan Grossman

Problem Statement

Given
- an energy consumption time series aggregated over multiple appliances
- profiles describing the amplitude, duration and frequency of energy consumption pulses for individual appliances

Find
- the energy consumption time series disaggregated for each of the individual appliances

The Data:

This is aggregate energy consumption data for one home that contains the following main appliances,

- Central AC 1 - The most common repeating pulse. (At about 2.5 KW amplitude and a width of about 10 minutes)
- Central AC 2 - Another repeating pulse but less frequent (At about 4 KW amplitude and > 30 minute width)
- Pool Pump - Runs for a duration of about 3 hours at 1.5 KW amplitude. Starts at the same time everyday.
- Refrigerator - This is the smallest amplitude repeating pulse at < 200 W

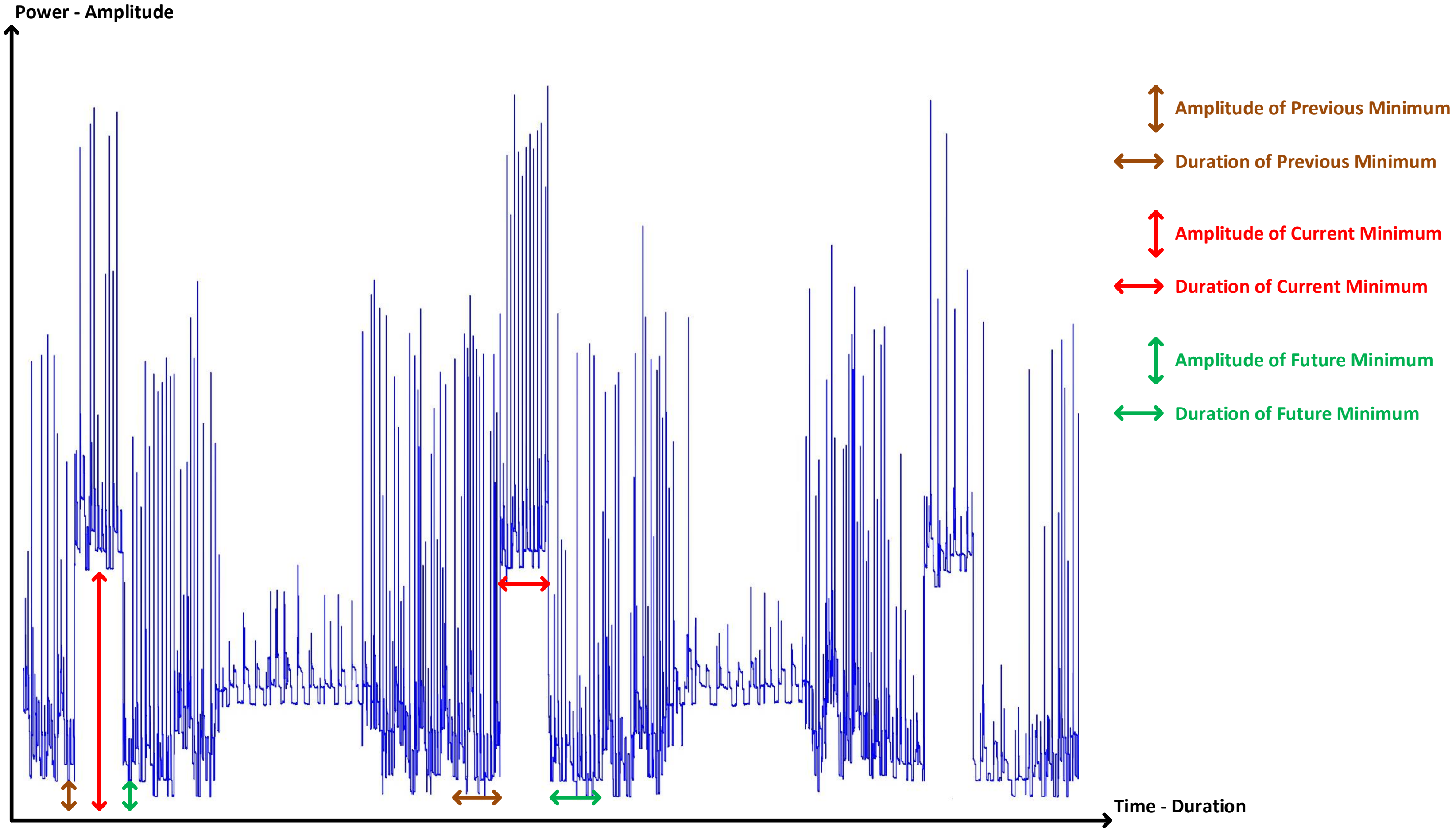
If you are plotting the time series you should be able to spot all of the above 4 quite easily.

The Challenge:

Process the above data to extract the energy consumption time series for individual appliances listed above.

Pulse Detection

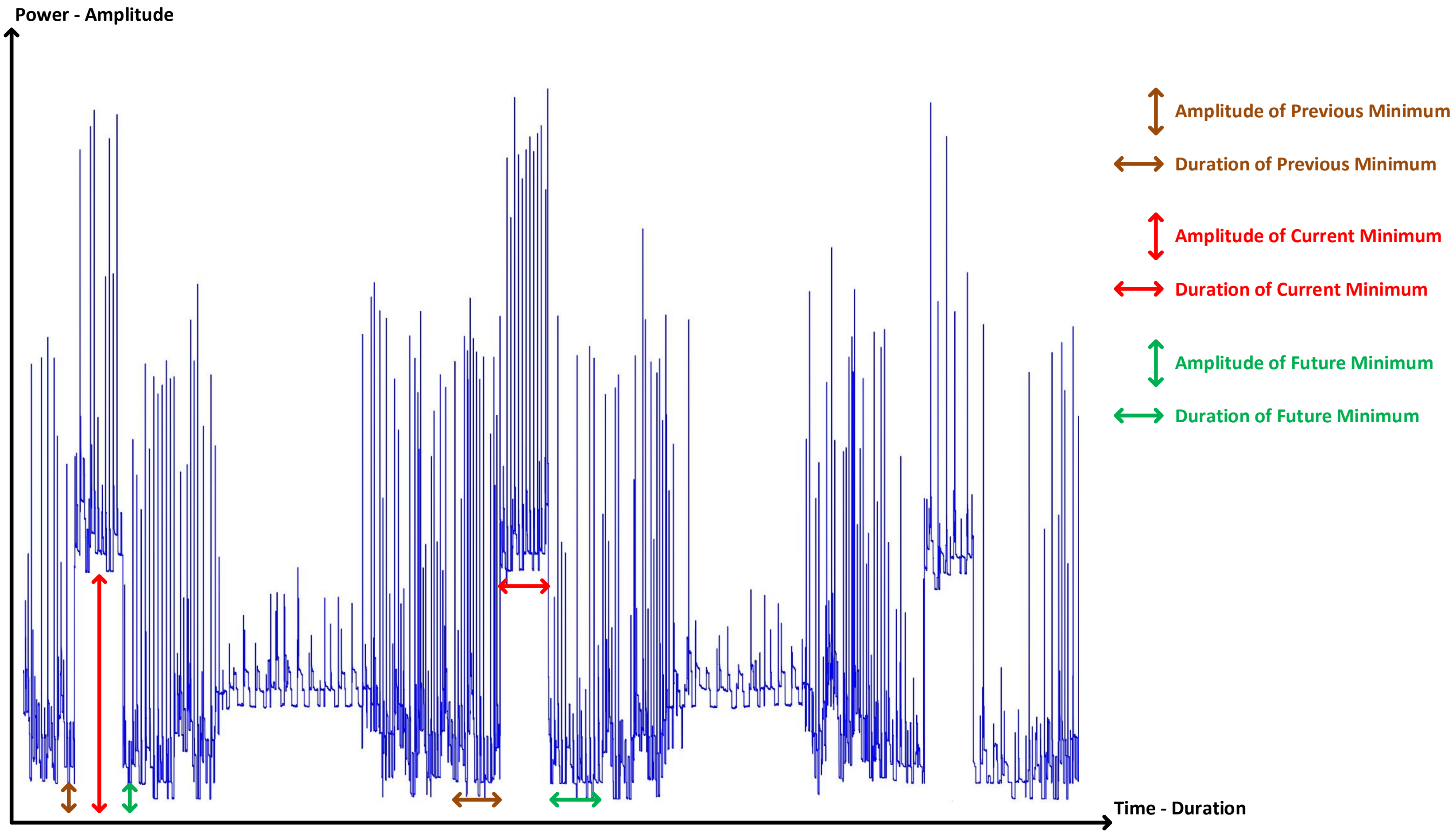
The energy consumption time series given is the aggregate of the individual time series for multiple appliances. Hence, for purposes of pulse detection, the quantity of interest is the minimum amplitude over a given interval—rather than the particular amplitude at a given time—since the pulse for a given appliance will set a floor value for the aggregate signal over the duration of the pulse.



Pulse Detection

A pulse is detected if

- the minimum for the current interval under consideration is greater than the minimum for the previous interval by an amount approximately equal to the expected height of the pulse
- the minimum for the future interval approaches to the minimum for the previous interval
- the duration of the current interval is approximately equal to the expected width of the pulse



Pulse Detection Implementation

The pulse detection algorithm is implemented in the code as a binary state machine in which

- the *pulseFound* state is entered when the pulse detection criteria (described on the previous slide) are met
- the *pulseFound* state is exited when the signal level decreases

```
def findPulses(aggregateSignal, expectedHeight, expectedWidth, heightMargin, widthMargin):
    pulseTrain = np.zeros(len(aggregateSignal))
    pulseFound = 0
    estimatedHeight = 0
    # For every sample in the time series under consideration ...
    for i in range(expectedWidth,(len(aggregateSignal) - int(2*expectedWidth))):
        # If not in the pulseFound state, check whether the conditions are met for entering the pulseFound state.
        if pulseFound == 0:
            previousMin = min(aggregateSignal[(i - int((widthMargin/8)*expectedWidth)):(i - 1)])
            currentMin = min(aggregateSignal[i:(i + int((1 - 2*widthMargin)*expectedWidth))])
            futureMin = min(aggregateSignal[(i + expectedWidth + 1):(i + expectedWidth + \
                                                                     int(4*widthMargin*expectedWidth))])
            # If the minimum of the time series over the "current and short-term future range" of samples is:
            # (1) greater than the minimum of the time series over the "past range" of samples (by an amount
            # approximately equal to the expected height of the pulse); and
            # (2) greater than the minimum of the time series over the "longer-term future range" of samples (by an
            # amount approximately equal to the expected height of the pulse); then
            # enter the pulseFound state.
            # Note that the "short-term future range" extends from the current instant to a future interval equal to
            # the approximate expected width of the pulse; and the "longer-term future range" extends into the future
            # beyond the approximate expected width of the pulse.
            if ((currentMin - previousMin) > (1 - 2*heightMargin)*expectedHeight) \
                & ((currentMin - previousMin) < (1 + 4*heightMargin)*expectedHeight) \
                & ((futureMin - previousMin) < (1 - 2*heightMargin)*expectedHeight):
                pulseFound = 1
                estimatedHeight = currentMin - previousMin
        # If in the pulseFound state, check whether the conditions are met for exiting the pulseFound state.
        if pulseFound == 1:
            # If the current sammple is less than the minimum of the time series over the approximate expected width
            # of the pulse, then exit the pulseFound state.
            if aggregateSignal[i] < currentMin:
                pulseFound = 0
                estimatedHeight = 0
        # Set the current sample of the estimated pulse train equal to the estimated height of the pulse,
        # which equal to:
        # the delta between "the current and short-term future minimum" and
        # "the past minimum" if in the pulseFound state; and
        # zero if not in the pulseFound state.
        pulseTrain[i] = estimatedHeight
    return pulseTrain
```

Energy Consumption Estimation

For each time instant, the current value of the energy consumption for a given appliance is estimated as

- the increase in the signal level at the start of the current pulse if currently in the *pulseFound* state
- zero if not currently in the *pulseFound* state

```
def findPulses(aggregateSignal, expectedHeight, expectedWidth, heightMargin, widthMargin):
    pulseTrain = np.zeros(len(aggregateSignal))
    pulseFound = 0
    estimatedHeight = 0
    # For every sample in the time series under consideration ...
    for i in range(expectedWidth,(len(aggregateSignal) - int(2*expectedWidth))):
        # If not in the pulseFound state, check whether the conditions are met for entering the pulseFound state.
        if pulseFound == 0:
            previousMin = min(aggregateSignal[(i - int((widthMargin/8)*expectedWidth)):(i - 1)])
            currentMin = min(aggregateSignal[i:(i + int((1 - 2*widthMargin)*expectedWidth))])
            futureMin = min(aggregateSignal[(i + expectedWidth + 1):(i + expectedWidth + \
                                                                  int(4*widthMargin*expectedWidth))])
            # If the minimum of the time series over the "current and short-term future range" of samples is:
            # (1) greater than the minimum of the time series over the "past range" of samples (by an amount
            # approximately equal to the expected height of the pulse); and
            # (2) greater than the minimum of the time series over the "longer-term future range" of samples (by an
            # amount approximately equal to the expected height of the pulse); then
            # enter the pulseFound state.
            # Note that the "short-term future range" extends from the current instant to a future interval equal to
            # the approximate expected width of the pulse; and the "longer-term future range" extends into the future
            # beyond the approximate expected width of the pulse.
            if ((currentMin - previousMin) > (1 - 2*heightMargin)*expectedHeight) \
                & ((currentMin - previousMin) < (1 + 4*heightMargin)*expectedHeight) \
                & ((futureMin - previousMin) < (1 - 2*heightMargin)*expectedHeight):
                pulseFound = 1
                estimatedHeight = currentMin - previousMin
        # If in the pulseFound state, check whether the conditions are met for exiting the pulseFound state.
        if pulseFound == 1:
            # If the current sammple is less than the minimum of the time series over the approximate expected width
            # of the pulse, then exit the pulseFound state.
            if aggregateSignal[i] < currentMin:
                pulseFound = 0
                estimatedHeight = 0
        # Set the current sample of the estimated pulse train equal to the estimated height of the pulse,
        # which equal to:
        # the delta between "the current and short-term future minimum" and
        # "the past minimum" if in the pulseFound state; and
        # zero if not in the pulseFound state.
        pulseTrain[i] = estimatedHeight
    return pulseTrain
```

Computational Cost

The basic building blocks of the pulse detection algorithm are comparisons, both explicit comparisons embedded in the if-statements and implicit comparisons embedded in the min-functions. Hence the computational cost can be represented as the total number of comparisons performed on each call to the *findPulses()* function as

$$\begin{aligned} N_{comparisons} &= N_0[0.125 m_w w_e + (1 - 2m_w) w_e + 4 m_w w_e + 3] + N_1[1] \\ &= N_0[(1 + 2.125 m_w) w_e + 3] + N_1 \end{aligned}$$

where

$$\begin{aligned} N_0 &= \mathit{the\ number\ of\ samples\ considered\ when\ not\ in\ the\ pulseFound\ state} \\ N_1 &= \mathit{the\ number\ of\ samples\ considered\ when\ in\ the\ pulseFound\ state} \\ m_w &= \mathit{the\ pulse\ width\ margin} \\ w_e &= \mathit{the\ expected\ pulse\ width} \end{aligned}$$

Appliance Profiles

The appliance profiles given were approximate and were refined by inspection of the data. Hence the profiles used in the code differ slightly from those given in the problem statement.

The Data:

This is aggregate energy consumption data for one home that contains the following main appliances,

- Central AC 1 - The most common repeating pulse. (At about 2.5 KW amplitude and a width of about 10 minutes)
- Central AC 2 - Another repeating pulse but less frequent (At about 4 KW amplitude and > 30 minute width)
- Pool Pump - Runs for a duration of about 3 hours at 1.5 KW amplitude. Starts at the same time everyday.
- Refrigerator - This is the smallest amplitude repeating pulse at < 200 W

```
##############################
# Define appliance profiles #
##############################
print "Defining appliance profiles"

ac1Profile = {'expectedHeight':2750, 'expectedWidth':int((12*samplesPerMinute)), \
              'heightMargin':0.10, 'widthMargin':0.15}

ac2Profile = {'expectedHeight':4000, 'expectedWidth':int((60*samplesPerMinute)), \
              'heightMargin':0.10, 'widthMargin':0.25}

pumpProfile = {'expectedHeight':1520, 'expectedWidth':int((2.8*samplesPerHour)), \
               'heightMargin':0.10, 'widthMargin':0.10}

fridgeProfile = {'expectedHeight':180, 'expectedWidth':int((45*samplesPerMinute)), \
                 'heightMargin':0.10, 'widthMargin':0.25}
```

Appliance Profiles

The profile given for the refrigerator was not only approximate, but also incomplete in the sense that the pulse duration was not given. Hence the duration of the refrigerator pulse was estimated both by inspection of the data and by review of the literature on appliance energy consumption profiles.

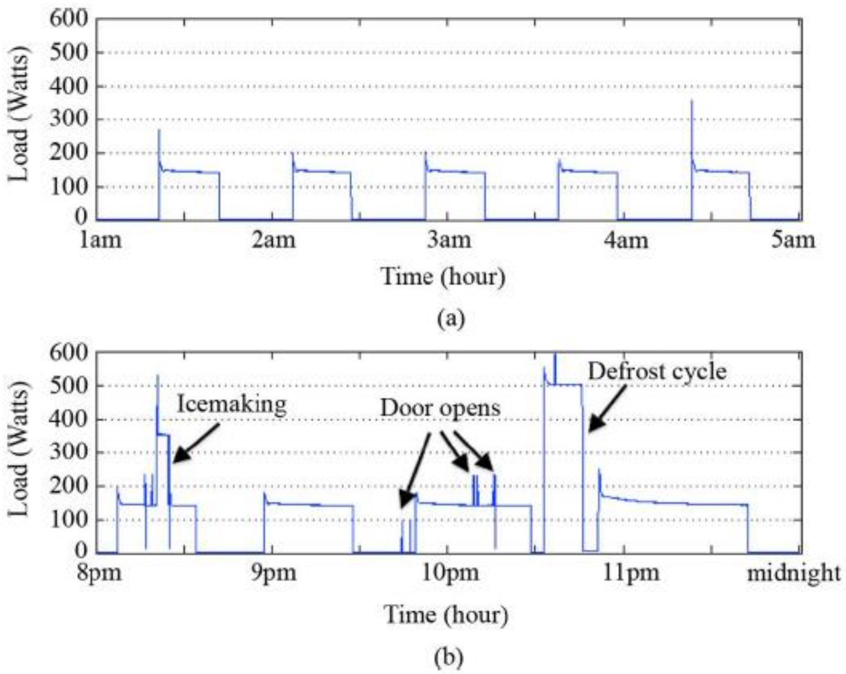
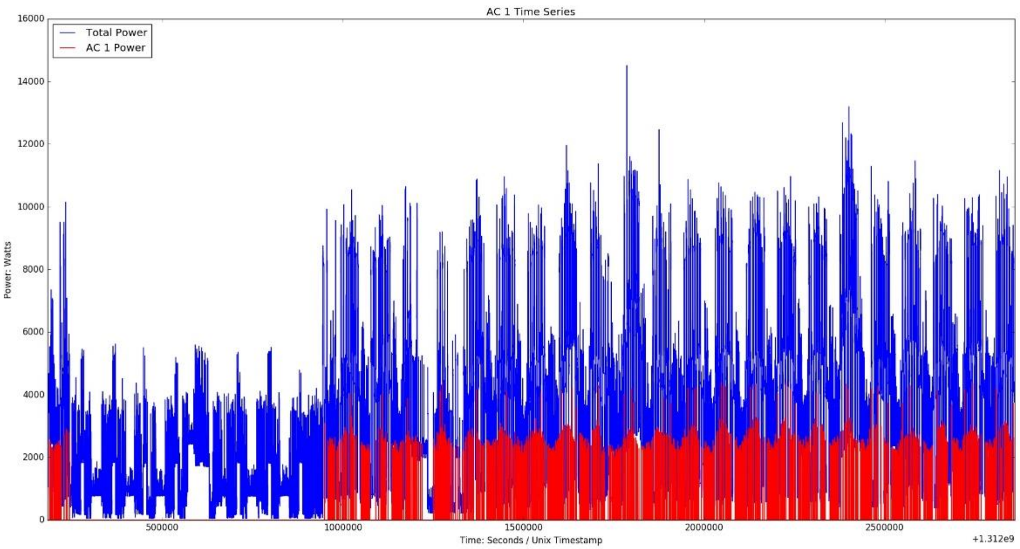


Fig. 12. Load profiles of the side-by-side refrigerator unit: (a) no activity; and (b) with door opens, an ice-making cycle and a defrost cycle.

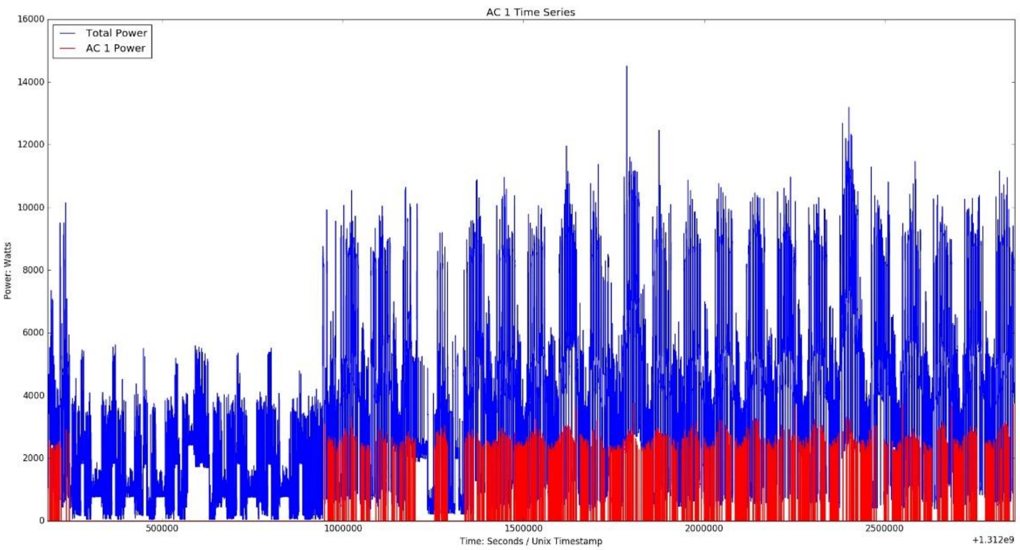
Load Profiles of Selected Major Household Appliances and Their Demand Response Opportunities, Pipattanasomporn, Kuzlu, Rahman and Teklu, **IEEE Transactions on Smart Grid**, Vol. 5, No. 2, March 2014

AC 1 Time Series

AC 1 Higher Margin and Lower Margin Profiles			
Expected Height	Expected Width	Height Margin	Width Margin
2750 W	12 min	0.15	0.20
2750 W	12 min	0.10	0.15



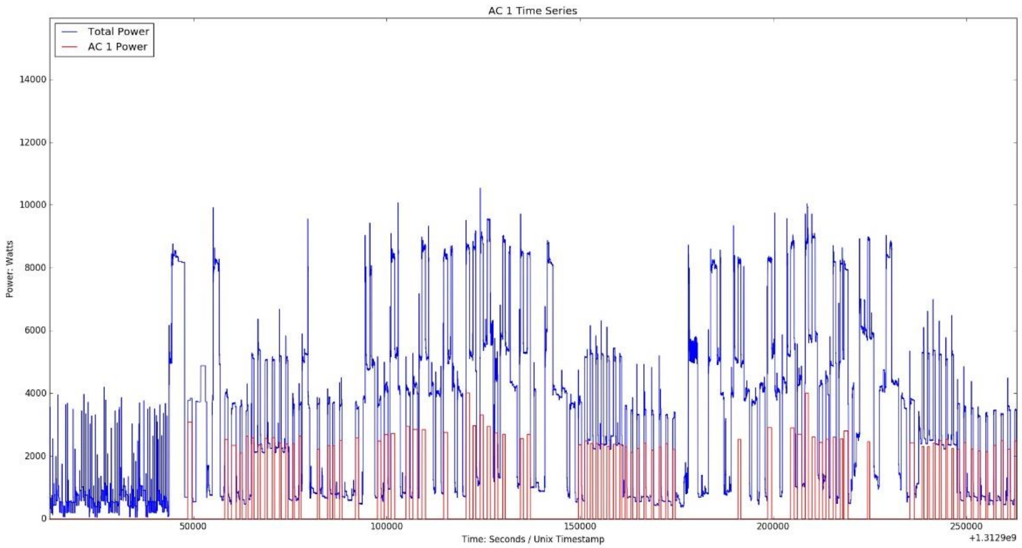
Higher Margins - Lower Missed Detection



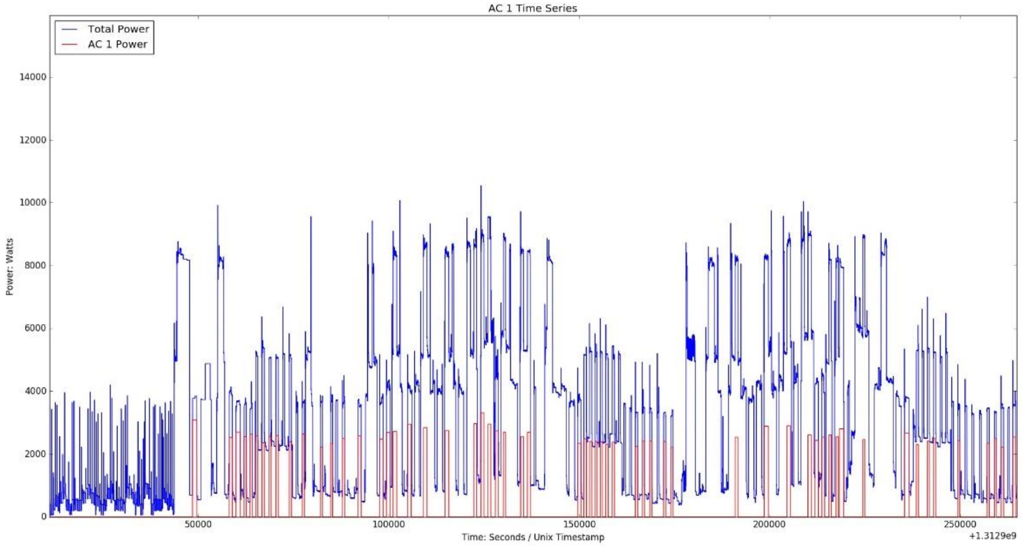
Lower Margins - Lower False Detection

AC 1 Time Series – Close Up

AC 1 Higher Margin and Lower Margin Profiles			
Expected Height	Expected Width	Height Margin	Width Margin
2750 W	12 min	0.15	0.20
2750 W	12 min	0.10	0.15



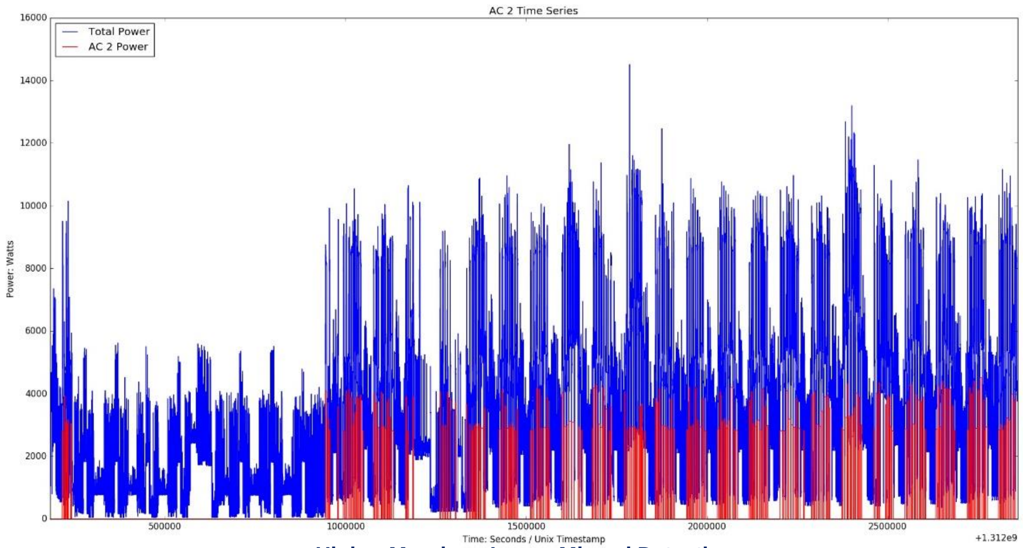
Higher Margins - Lower Missed Detection



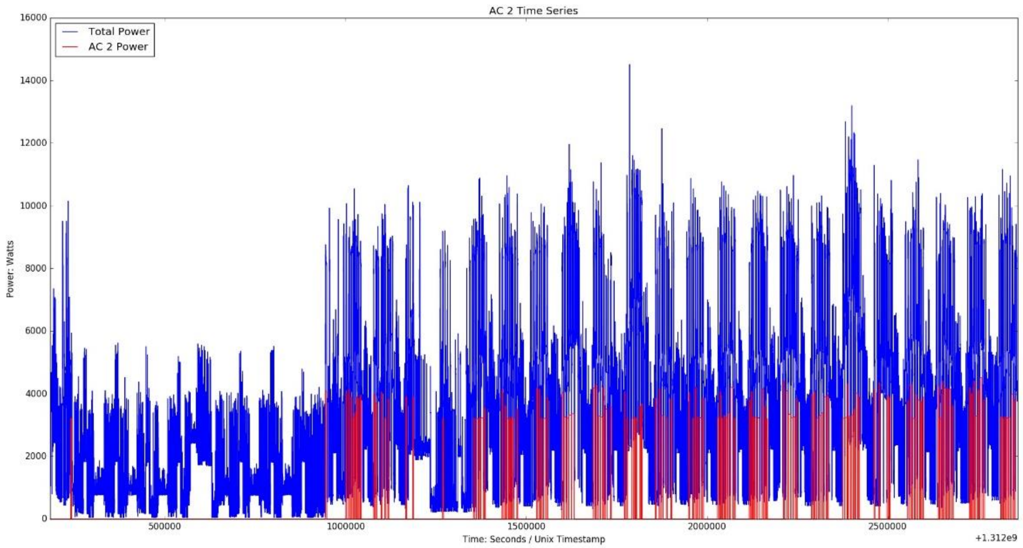
Lower Margins - Lower False Detection

AC 2 Time Series

AC 2 Higher Margin and Lower Margin Profiles			
Expected Height	Expected Width	Height Margin	Width Margin
4000 W	60 min	0.15	0.30
4000 W	60 min	0.10	0.25



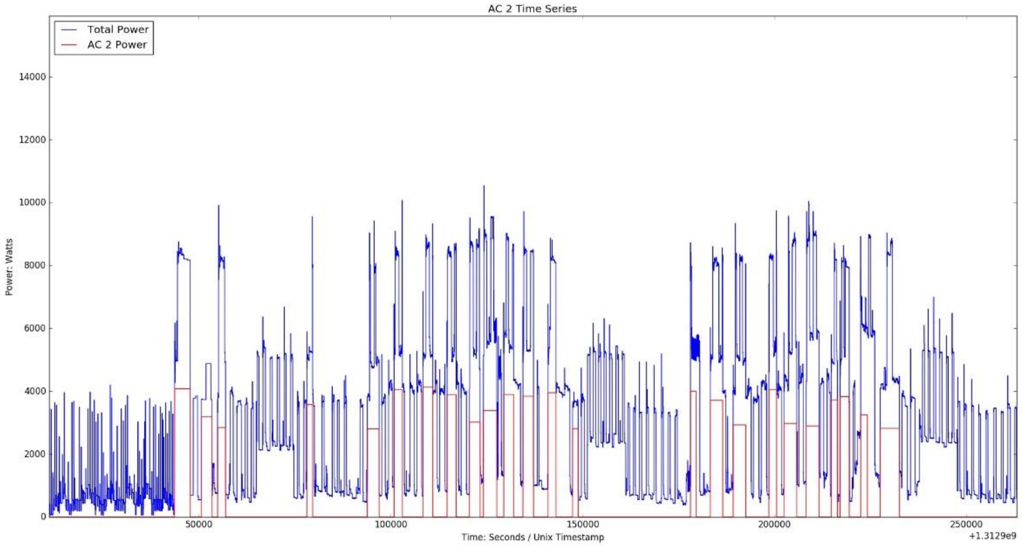
Higher Margins - Lower Missed Detection



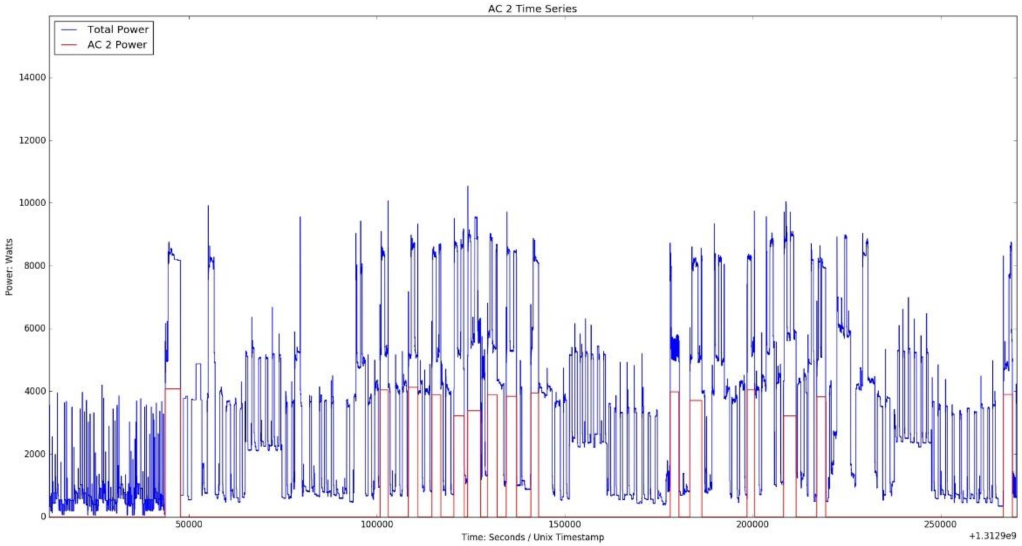
Lower Margins - Lower False Detection

AC 2 Time Series – Close Up

AC 2 Higher Margin and Lower Margin Profiles			
Expected Height	Expected Width	Height Margin	Width Margin
4000 W	60 min	0.15	0.30
4000 W	60 min	0.10	0.25



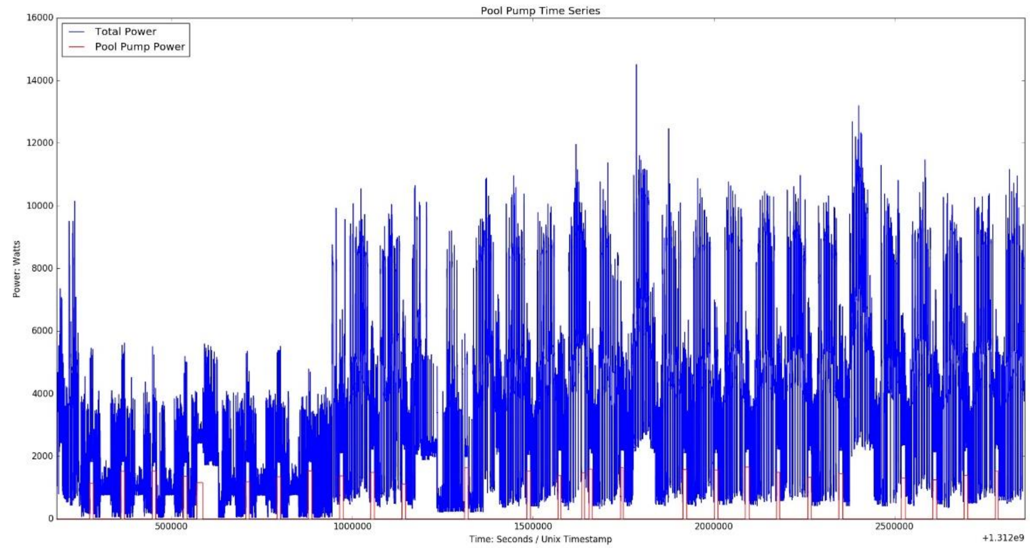
Higher Margins - Lower Missed Detection



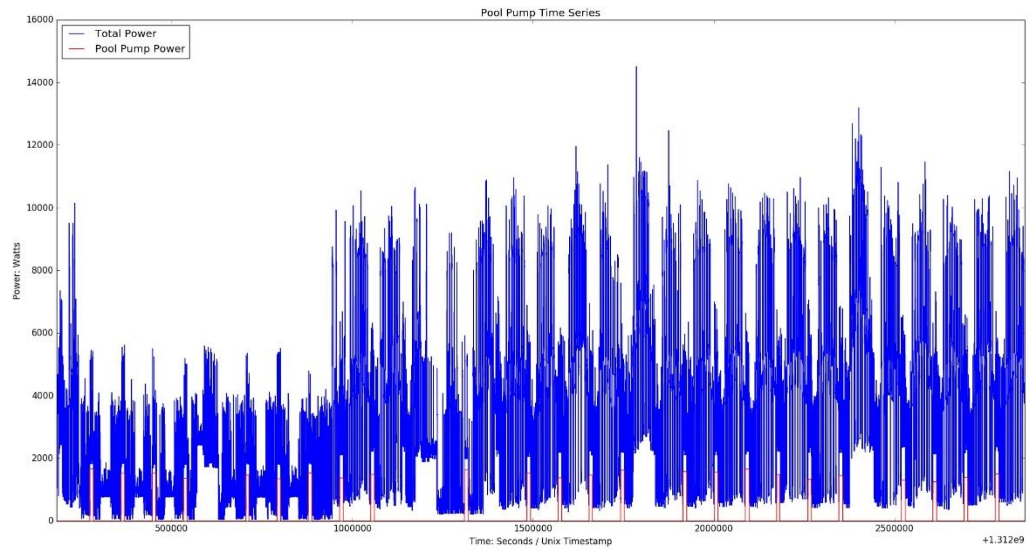
Lower Margins - Lower False Detection

Pool Pump Time Series

Pool Pump Higher Margin and Lower Margin Profiles			
Expected Height	Expected Width	Height Margin	Width Margin
1520 W	2.8 hrs	0.15	0.15
1520 W	2.8 hrs	0.10	0.10



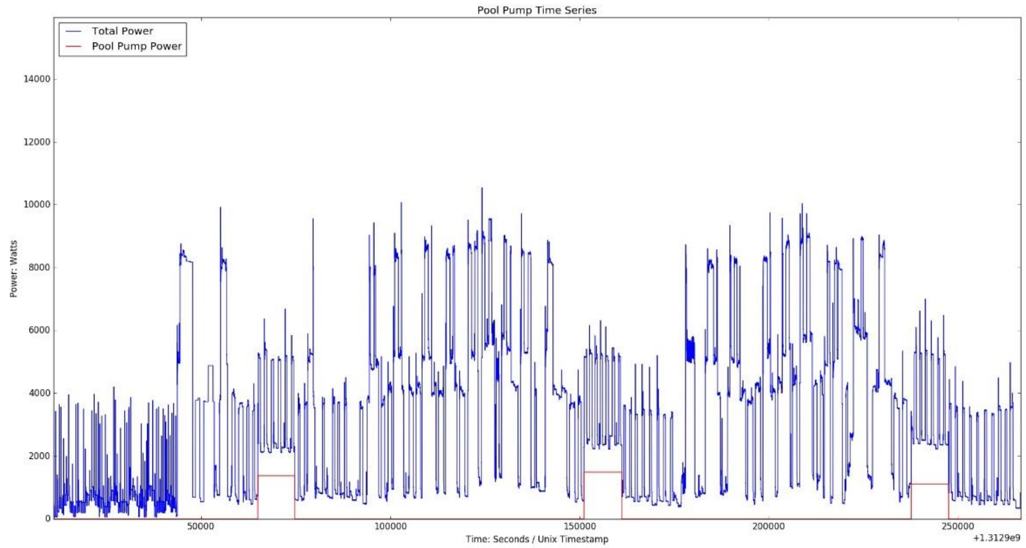
Higher Margins - Lower Missed Detection



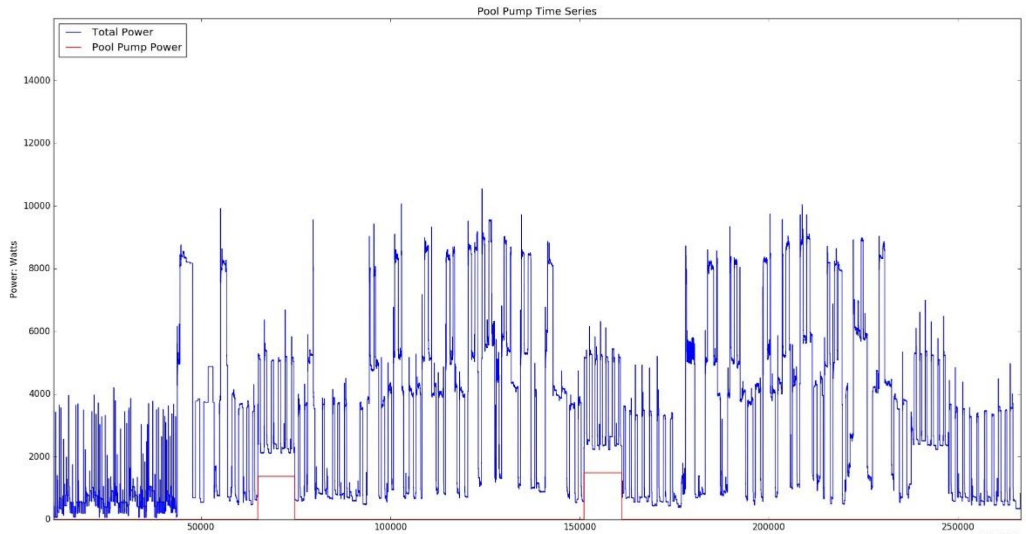
Lower Margins - Lower False Detection

Pool Pump Time Series – Close Up

Pool Pump Higher Margin and Lower Margin Profiles			
Expected Height	Expected Width	Height Margin	Width Margin
1520 W	2.8 hrs	0.15	0.15
1520 W	2.8 hrs	0.10	0.10



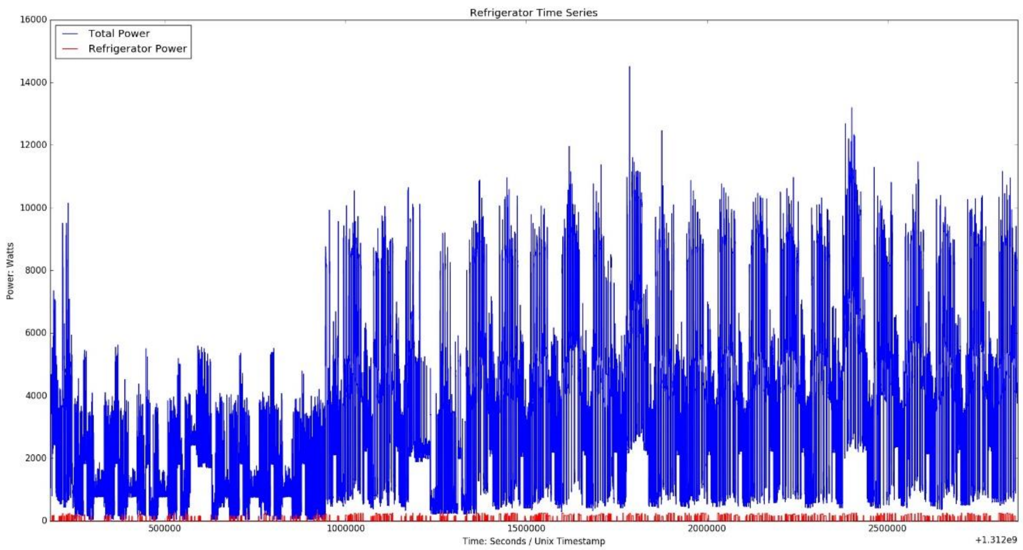
Higher Margins - Lower Missed Detection



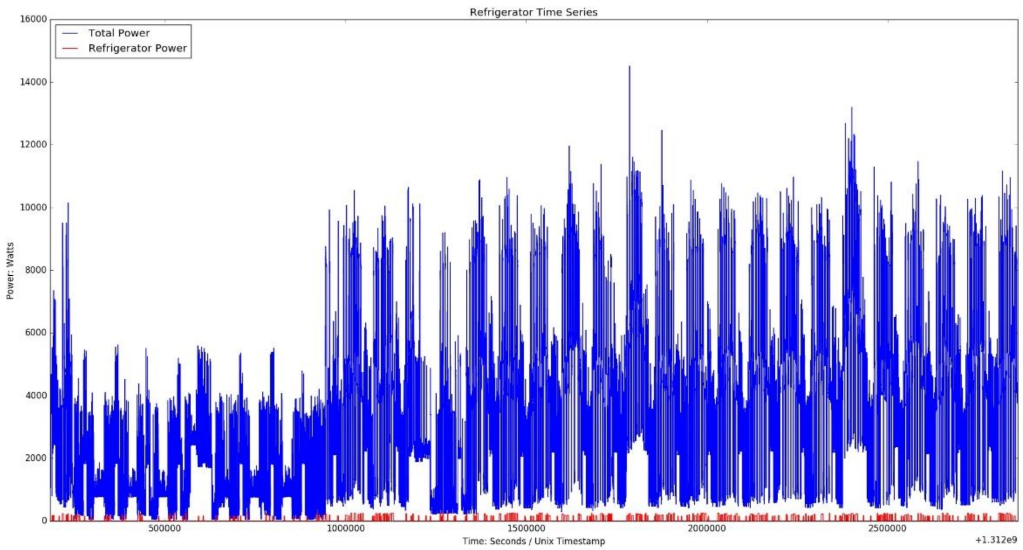
Lower Margins - Lower False Detection

Refrigerator Time Series

Refrigerator Higher Margin and Lower Margin Profiles			
Expected Height	Expected Width	Height Margin	Width Margin
180 W	45 min	0.10	0.30
180 W	45 min	0.10	0.25



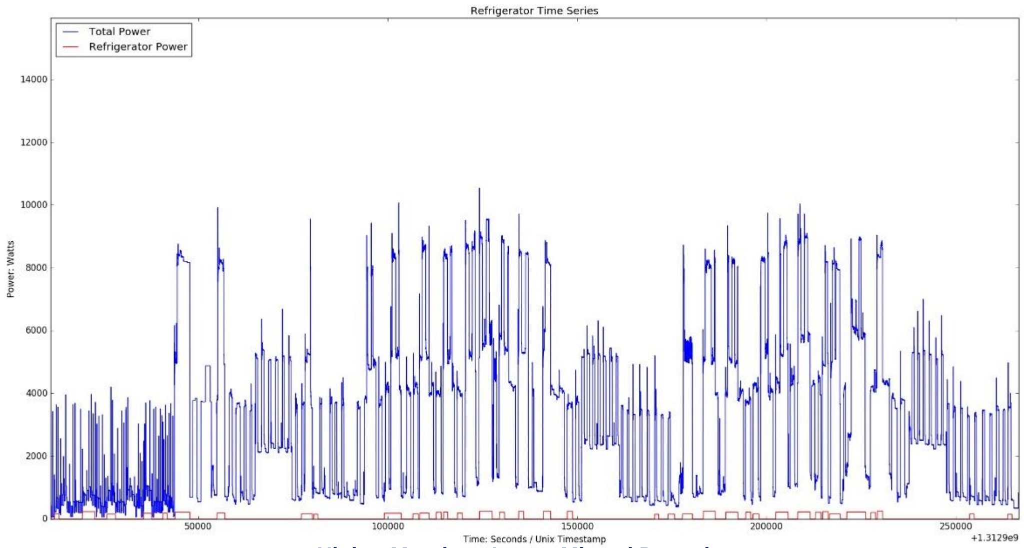
Higher Margins - Lower Missed Detection



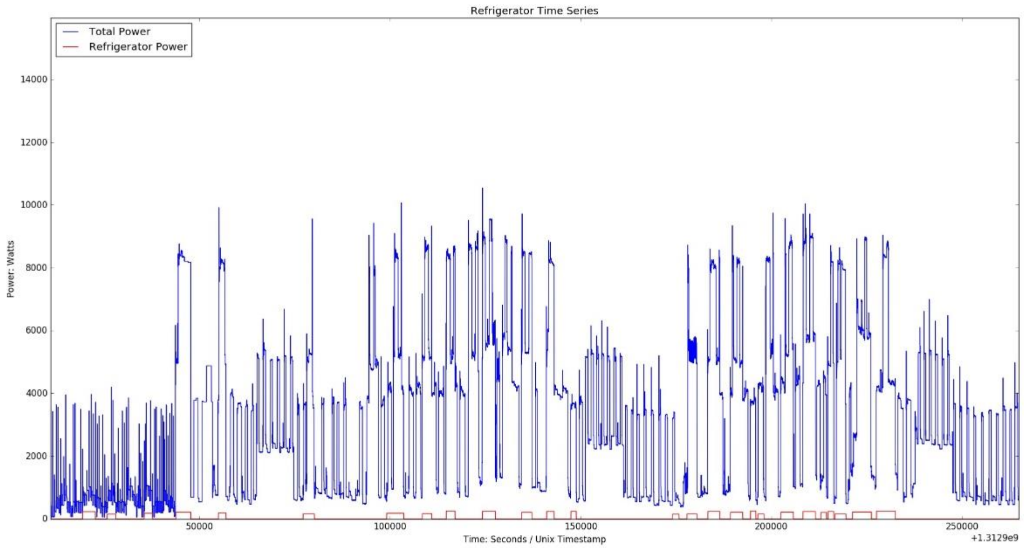
Lower Margins - Lower False Detection

Refrigerator Time Series – Close Up

Refrigerator Higher Margin and Lower Margin Profiles			
Expected Height	Expected Width	Height Margin	Width Margin
180 W	45 min	0.10	0.30
180 W	45 min	0.10	0.25



Higher Margins - Lower Missed Detection



Lower Margins - Lower False Detection

Potential Solutions without Appliance Profiles

If appliance profiles were not given, energy consumption disaggregation could be attempted after estimating appliance profiles by

- visual inspection of the data to recognize recurring pulses and then characterize them by amplitude, duration, frequency, etc.
- brute force detection of pulses of varying amplitude and duration, followed by clustering the detected pulses according to amplitude and duration, or equivalently binning the detected pulses according to ranges of amplitude and duration to create two-dimensional histograms
- correlation of recognized clusters or categories of pulses with known appliance types and profiles, either from a proprietary database thereof, or from publically available literature thereon

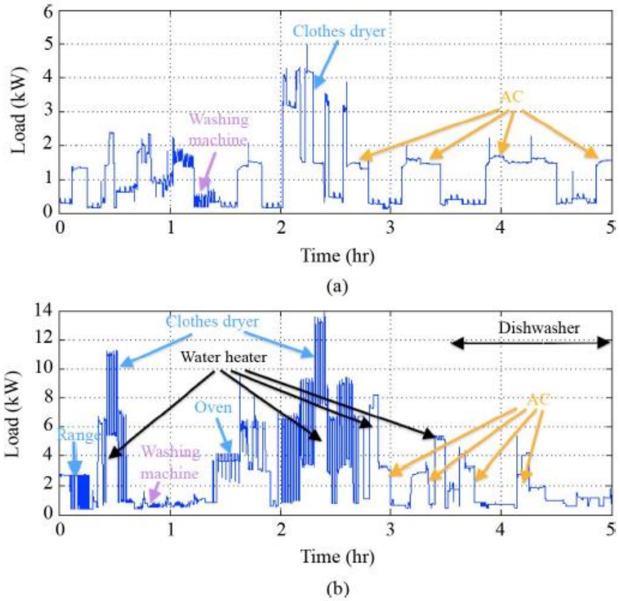


Fig. 1. Household load profile (kW) between 11am and 4pm: (a) House 1: 1200 sq ft; and (b) House 2: 2500 sq ft.

TABLE I
APPLIANCE MODELS AND THEIR RATINGS

Appliance	Model	Rating
Clothes washer	- GE WSM2420D3WW	120/240V 21A
	- LG WM2016CW	120V 60Hz 5A
Clothes dryer	- GE WSM2420D3WW	120/208V 60Hz 16A
	- LG DLE2516W	120/240V 60Hz 26A
Air conditioner	- LG LW1212ER	115V 12,000 BTU
	- Bryant Heating and Cooling Systems 697CN030-B	208/230V 1PH 60Hz Comp: RLA 14.2A Fan: 1/5HP
Water heater	- E52-50R-045DV (50 gal)	240V 4500W
Range/oven	- Kenmore 790.91312013	120/240V 10kW
Dishwasher	- Kenmore 665.13242K900	120V 60Hz 9.6A
Refrigerator	- Hotpoint HTR16ABSRWW	120V 60Hz 12A 15.6-cuft top freezer
	- Maytag MSD2641KEW	115V 60Hz 9.4A 25.6-cuft side-by-side

Load Profiles of Selected Major Household Appliances and Their Demand Response Opportunities, Pipattanasomporn, Kuzlu, Rahman and Teklu, **IEEE Transactions on Smart Grid**, Vol. 5, No. 2, March 2014

Thank You
for
Your Time

Appendix


```python
from __future__ import division
from math import *
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

def findPulses(aggregateSignal, expectedHeight, expectedWidth, heightMargin, widthMargin):
    pulseTrain = np.zeros(len(aggregateSignal))
    pulseFound = 0
    estimatedHeight = 0
    # For every sample in the time series under consideration ...
    for i in range(expectedWidth,(len(aggregateSignal) - int(2*expectedWidth))):
        # If not in the pulseFound state, check whether the conditions are met for entering the pulseFound state.
        if pulseFound == 0:
            previousMin = min(aggregateSignal[(i - int((widthMargin/8)*expectedWidth)):(i - 1)])
            currentMin = min(aggregateSignal[i:(i + int((1 - 2*widthMargin)*expectedWidth))])
            futureMin = min(aggregateSignal[(i + expectedWidth + 1):(i + expectedWidth + \
                                                        int(4*widthMargin*expectedWidth))])
            # If the minimum of the time series over the "current and short-term future range" of samples is:
            # (1) greater than the minimum of the time series over the "past range" of samples (by an amount
            # approximately equal to the expected height of the pulse); and
            # (2) greater than the minimum of the time series over the "longer-term future range" of samples (by an
            # amount approximately equal to the expected height of the pulse); then
            # enter the pulseFound state.
            # Note that the "short-term future range" extends from the current instant to a future interval equal to
            # the approximate expected width of the pulse; and the "longer-term future range" extends into the future
            # beyond the approximate expected width of the pulse.
            if ((currentMin - previousMin) > (1 - 2*heightMargin)*expectedHeight) \
                & ((currentMin - previousMin) < (1 + 4*heightMargin)*expectedHeight) \
                & ((futureMin - previousMin) < (1 - 2*heightMargin)*expectedHeight):
                pulseFound = 1
                estimatedHeight = currentMin - previousMin
        # If in the pulseFound state, check whether the conditions are met for exiting the pulseFound state.
        if pulseFound == 1:
            # If the current sammple is less than the minimum of the time series over the approximate expected width
            # of the pulse, then exit the pulseFound state.
            if aggregateSignal[i] < currentMin:
                pulseFound = 0
                estimatedHeight = 0
        # Set the current sample of the estimated pulse train equal to the estimated height of the pulse,
        # which equal to:
        # the delta between "the current and short-term future minimum" and
        # "the past minimum" if in the pulseFound state; and
        # zero if not in the pulseFound state.
        pulseTrain[i] = estimatedHeight
    return pulseTrain


print "Begin Energy Consumption Disaggregation"

###########################
# Read data from CSV file #
###########################
print "Reading data from CSV file"

data = pd.read_csv('data.csv', names=['time', 'power'])
time = np.asarray(data['time'])
```

Appendix

```
energy_consumption_disaggregation.py                                              Page 2

power = np.asarray(data['power'])

#############################################################
# Specify number of samples for intervals of various widths #
#############################################################
print "Specifying number of samples for intervals of various widths"
samplesPerSecond = 1
samplesPerMinute = 60 * samplesPerSecond
samplesPerHour = 60 * samplesPerMinute
samplesPerDay = 24 * samplesPerHour
samplesPerMonth = 31 * samplesPerDay

#############################
# Define appliance profiles #
#############################
print "Defining appliance profiles"

ac1Profile = {'expectedHeight':2750, 'expectedWidth':int((12*samplesPerMinute)), \
              'heightMargin':0.10, 'widthMargin':0.15}

ac2Profile = {'expectedHeight':4000, 'expectedWidth':int((60*samplesPerMinute)), \
              'heightMargin':0.10, 'widthMargin':0.25}

pumpProfile = {'expectedHeight':1520, 'expectedWidth':int((2.8*samplesPerHour)), \
               'heightMargin':0.10, 'widthMargin':0.10}

fridgeProfile = {'expectedHeight':180, 'expectedWidth':int((45*samplesPerMinute)), \
                 'heightMargin':0.10, 'widthMargin':0.25}

testSignal = power
# startPoint = 1600000
# endPoint = startPoint + 3 * samplesPerDay
# testSignal = power[startPoint:(endPoint)]
# power = power[startPoint:(endPoint)]
# time = time[startPoint:(endPoint)]

##########################
# Search for AC 1 pulses #
##########################
print "Searching for AC 1 pulses"

ac1Signal = findPulses(testSignal, ac1Profile['expectedHeight'], ac1Profile['expectedWidth'], \
                       ac1Profile['heightMargin'], ac1Profile['widthMargin'])

print "Writing AC 1 data to CSV file"

ac1Columns = ['time', 'power', 'ac1Signal']
ac1Data = pd.DataFrame([], columns=list(ac1Columns))
ac1Data['time'] = time
ac1Data['power'] = power
ac1Data['ac1Signal'] = ac1Signal
ac1Data.to_csv('ac1_signal_data.csv')

##########################
# Search for AC 2 pulses #
##########################
print "Searching for AC 2 pulses"
```

Appendix

energy_consumption_disaggregation.py   Page 3

```
ac2Signal = findPulses(testSignal, ac2Profile['expectedHeight'], ac2Profile['expectedWidth'], \
                        ac2Profile['heightMargin'], ac2Profile['widthMargin'])

print "Writing AC 2 data to CSV file"

ac2Columns = ['time', 'power', 'ac2Signal']
ac2Data = pd.DataFrame([], columns=list(ac2Columns))
ac2Data['time'] = time
ac2Data['power'] = power
ac2Data['ac2Signal'] = ac2Signal
ac2Data.to_csv('ac2_signal_data.csv')

###############################
# Search for pool pump pulses #
###############################
print "Searching for pool pump pulses"

pumpSignal = findPulses(testSignal, pumpProfile['expectedHeight'], pumpProfile['expectedWidth'], \
                        pumpProfile['heightMargin'], pumpProfile['widthMargin'])

print "Writing pool pump data to CSV file"

pump1Columns = ['time', 'power', 'pumpSignal']
pump1Data = pd.DataFrame([], columns=list(pump1Columns))
pump1Data['time'] = time
pump1Data['power'] = power
pump1Data['pumpSignal'] = pumpSignal
pump1Data.to_csv('pump_signal_data.csv')

##################################
# Search for refrigerator pulses #
##################################
print "Searching for refrigerator pulses"

fridgeSignal = findPulses(testSignal, fridgeProfile['expectedHeight'], fridgeProfile['expectedWidth'], \
                          fridgeProfile['heightMargin'], fridgeProfile['widthMargin'])

print "Writing refrigerator data to CSV file"

fridge1Columns = ['time', 'power', 'fridgeSignal']
fridge1Data = pd.DataFrame([], columns=list(fridge1Columns))
fridge1Data['time'] = time
fridge1Data['power'] = power
fridge1Data['fridgeSignal'] = fridgeSignal
fridge1Data.to_csv('fridge_signal_data.csv')

###############################################
# Read and plot processed data from CSV files #
###############################################
print "Reading processed data from CSV file"

# Read AC 1 data
ac1Data = pd.read_csv('ac1_signal_data.csv')
time = np.asarray(ac1Data['time'])
power = np.asarray(ac1Data['power'])
ac1Signal = np.asarray(ac1Data['ac1Signal'])

# Plot AC 1 data
```

Appendix


```python
plt.plot(time, power, label='Total Power', color='blue')
plt.plot(time, ac1Signal, label='AC 1 Power', color='red')
plt.legend(loc='upper left')
plt.xlim(min(time), max(time))
plt.xlabel("Time: Seconds / Unix Timestamp")
plt.ylabel("Power: Watts")
plt.title("AC 1 Time Series")
plt.show()

# Read AC 2 data
ac2Data = pd.read_csv('ac2_signal_data.csv')
time = np.asarray(ac2Data['time'])
power = np.asarray(ac2Data['power'])
ac2Signal = np.asarray(ac2Data['ac2Signal'])

# Plot AC 2 data
plt.plot(time, power, label='Total Power', color='blue')
plt.plot(time, ac2Signal, label='AC 2 Power', color='red')
plt.legend(loc='upper left')
plt.xlim(min(time), max(time))
plt.xlabel("Time: Seconds / Unix Timestamp")
plt.ylabel("Power: Watts")
plt.title("AC 2 Time Series")
plt.show()

# Read pool pump data
pumpData = pd.read_csv('pump_signal_data.csv')
time = np.asarray(pumpData['time'])
power = np.asarray(pumpData['power'])
pumpSignal = np.asarray(pumpData['pumpSignal'])

# Plot pool pump data
plt.plot(time, power, label='Total Power', color='blue')
plt.plot(time, pumpSignal, label='Pool Pump Power', color='red')
plt.legend(loc='upper left')
plt.xlim(min(time), max(time))
plt.xlabel("Time: Seconds / Unix Timestamp")
plt.ylabel("Power: Watts")
plt.title("Pool Pump Time Series")
plt.show()

# Read refrigerator data
fridgeData = pd.read_csv('fridge_signal_data.csv')
time = np.asarray(fridgeData['time'])
power = np.asarray(fridgeData['power'])
fridgeSignal = np.asarray(fridgeData['fridgeSignal'])

# Plot refrigerator data
plt.plot(time, power, label='Total Power', color='blue')
plt.plot(time, fridgeSignal, label='Refrigerator Power', color='red')
plt.legend(loc='upper left')
plt.xlim(min(time), max(time))
plt.xlabel("Time: Seconds / Unix Timestamp")
plt.ylabel("Power: Watts")
plt.title("Refrigerator Time Series")
plt.show()

print "End Energy Consumption Disaggregation"
```