UNIVERSITY OF TARTU

Institute of Computer Science

Computer Science Curriculum

Martin Jürgel

Non-Intrusive Load Monitoring in the OpenHAB smart home framework

Bachelor's Thesis (9 ECTS)

Supervisor: Jakob Mass, MSc

Tartu 2019

Elektrienergia disagregeerimine targa kodu raamistikus OpenHAB

Lühikokkuvõte:

Non-intrusive load monitoring (NILM) on elektrienergia tarbimise jälgimise meetod, milles rakendatakse masinõppe meetodeid, et automaatselt vooluvõrku ühendatud seadmete kogutarbest eraldada üksikute seadmete energiatarve. Käesolev bakalaureusetöö kirjeldab reaalajas töötavat NILM lahendust, mis saab sisendina kasutatavad energiatarbimise andmed targa kodu automatiseerimise keskkonnast openHAB. Kogu energiatarbimise disagregeerimiseks kasutatakse vabavaralist tööriista NILMTK, mis võimaldab kogu süsteemi energiatarbimisest üksikute seadmete tarbimist eraldada. Tuvastatud seadmete hetketarbimised saadetakse tagasi openHABi keskkonda, kus neid võib kasutada koduautomaatikas. Arvuiteaduse instituudi värkvõrgu laboris läbi viidud testide tulemused näitavad, et lahendus suudab täpselt tuvastada stabiilse ja suure energiatarbimisega seadmeid (näiteks soojapuhur), kuid on ebatäpne selliste seadmete eristamisel, mille energiatarbimine kõigub palju (näiteks kohvimasin) või moodustab kogutarbimisest vaid väikse osa (näiteks pirn).

Võtmesõnad:

Elektrienergia disagregeerimine, Asjade internet, Tark kodu, Energiatarbimise jälgimine, openHAB, Masinõpe, NILMTK

CERCS: P175

Non-Intrusive Load Monitoring in the OpenHAB smart home framework

Abstract:

Non-intrusive load monitoring (NILM) is an approach to energy monitoring, where machine learning techniques are applied to data from a single energy meter to determine the energy consumption of each appliance connected to the local electric network. This thesis presents a real-time NILM solution for the smart home that relies on the home automation platform openHAB for live power readings. NILMTK – an open-source energy disaggregation toolkit – is used to break down the aggregate live energy consumption data to the appliance level in real time. The disaggregated power data are then sent back to openHAB, where they can be displayed to the user and enable further automation. Tests conducted at the University of Tartu Internet of Things and Smart Solutions laboratory indicate high recognition accuracy for appliances with a steady high energy demand (e.g. a space heater), while lower accuracy scores were reported for appliances with a fluctuating power demand (e.g. a coffee maker) and low-powered appliances that only make up a small proportion of the total energy consumption (e.g. a light bulb).

Keywords:
Energy disaggregation, Internet of Things, Smart home, Energy monitoring, openHAB, Machine learning, NILMTK

CERCS: P175

Table of Contents

1 Introduction

A smart home integrates a number of interconnected devices such as sensors, controllers, actuators, etc., all of which are in constant communication with each other. Such networks of devices – also known as Internet of Things (IoT) networks – can be leveraged to make a positive impact on the energy efficiency of homes [1]. Intelligent, real-time decision making based on data from sensors can help lower energy consumption, thereby reducing a home's economic footprint and saving expenses.

Historically, residential energy monitoring has mostly been conducted by electricity suppliers to keep track of consumers' electricity usage to bill them accordingly. However, with the ever-growing increase in power demand [2] and the negative effect electricity generation has on the environment, it is important to find new methods of saving energy. Several countries have started setting up *smart grids* to achieve better energy efficiency and distribution automation. Smart meters are an important component of smart grids. As of September 2018, more than 12.8 million smart meters were operating in homes and small businesses across the United Kingdom [3]. The widespread use of smart meters that can be integrated into a smart home network not only makes it more convenient for consumers to get an overview of their total energy consumption at any given time, but also enables services such as real-time energy consumption feedback at the appliance level. According to Ehrhardt-Martinez et al. (2010), such feedback can lead to energy savings of up to 12% [4].

1.1 Motivating Scenario

Consider the following hypothetical scenario: Sam is running late for an important meeting. While on the bus, Sam realizes that – in his haste – he forgot to check if he unplugged the clothes iron. Fortunately, Sam's home is equipped with a real-time energy monitoring system that can be accessed remotely via his phone or any similar devices.

This energy monitoring system uses a single smart meter that monitors the entire house's electricity consumption in real time. The consumption data are sent to a central controller in Sam's home, where the data are analyzed and an appliance-level breakdown of the power consumption data is displayed on his home automation platform. Instead of going back home to check and risk missing the meeting, Sam can now simply connect to his smart home via his smartphone and view the appliance-level power consumption data and disable the clothes iron if found to be active.

For systems such as the one described above to work efficiently, accurate load monitoring is crucial. There are two approaches to monitoring energy consumption in smart homes [5]:

- Intrusive Load Monitoring (ILM), where multiple sub-meters are required to measure load at the appliance level.
- Non-instrusive Load Monitoring (NILM), where only overall consumption data across multiple appliances are provided by a single meter. Machine learning techniques are then applied to estimate the power consumed by each appliance connected to the local electric network. This process is also referred to as load disaggregation.

While approaches using ILM provide better accuracy since no estimating is required, setting up meters for individual appliances is impractical and not cost effective for most smart homes.

1.2 Aim of the Thesis

The aim of this thesis is to create a solution that is able to determine the operating states and estimate the real-time power consumption of appliances connected to the local electric network in a home environment using real-time data from a single meter. To achieve this, NILM and data analysis approaches are applied to disaggregate the data provided by the meter. The NILM output is then forwarded to the open-source smart home platform openHAB, which in addition to displaying the data, enables rule-based automation and other integration based on the consumption data (e.g. smart relays to turn devices on and off, occupancy detection, etc.). To achieve this, the contirbutions of this thesis are:

- First, a dataset consisting of appliance-level electricity consumption data in the test environment at the Internet of Things and Smart Solutions Lab is created.
- Next, an open-source toolkit that allows the comparison of energy disaggregation algorithms is used to find the most accurate algorithm on our dataset. This algorithm is then used to disaggregate power consumption data in real time.
- Finally, the real-time disaggregator is made compatible with openHAB – a home automation platform. Aggregate power readings are queried from openHAB and the disaggregated data sent back to openHAB to be displayed to the user in real time.

1.3 Outline

The structure of the thesis is as follows:

- In section 2, background information on the technologies used in this thesis is presented.

- In section 3, the architecture of the solution built as part of this work is presented and the building process described.
- In section 4, the accuracy of the created solution is evaluated.
- In section 5, a summary of the work is presented and suggestions for future research given.

2 Background

This chapter gives background information on non-intrusive load monitoring, but also introduces the technologies used in the practical portion of this thesis. Energy monitoring and machine learning are briefly covered, followed by a more in-depth look at NILM. Next, an open-source non-intrusive load monitoring toolkit is introduced and finally, an overview is presented of the home automation platform that is used later in the thesis.

2.1 Energy Monitoring in the Home

Power meters are a key component of any NILM system. There are several factors that need to be considered when setting up a home energy monitoring system. Different load disaggregation techniques require different measurement data sampled at different rates. While low-frequency real power readings provided by smart meters are sufficient for some approaches to NILM [6,7], other approaches that use features such as transient signal waveforms require data sampled at a rate of 1.2 kHz or higher [8].

2.1.1 Open Energy Monitor

The Open Energy Monitor project offers a number of open-source energy monitoring tools. It is targeted at people interested in a DIY approach to energy monitoring and includes a set of learning materials, which cover concepts like energy monitoring and the basics of AC power [9], sustainable energy [10], but also more in-depth tutorials such as how to build a breadboard-based home energy monitor to be used with an Arduino [11]. Also included is a web application for processing and visualising environmental data [12].

2.2 Machine Learning

The following paragraph is based on [13]. Machine learning is a subfield of computer science, where systems capable of learning from data without explicit instructions are constructed and studied. During the learning (training) process, a set of training data is analyzed and patterns and connections in the data are identified. As a result of this process, a model capable of finding similar connections in previously unseen data is created.

This is why approaches relying on machine learning are widely used in the field of non-intrusive load monitoring. Different buildings and appliances have different power signatures that are hard if not impossible to define by manually creating explicit rules for every possible scenario. Instead, a machine learning algorithm can be used to learn from prior power consumption data and automatically build a model that uses connections found in the data that

a human might not have even thought of. Machine learning algorithms can be broken down into two categories: supervised learning algorithms and unsupervised learning algorithms.

2.2.1 Learning Methods

In supervised learning, the training data are organized into pairs, which consist of an input value and its corresponding expected output value [13]. In the context of non-intrusive load monitoring, an example of such a labeled data set is a data set consisting of aggregate power consumption measurements (inputs) and their appliance-level breakdowns (desired outputs). A learning algorithm analyses the training data and generates a model that labels similar previously unseen data.

In unsupervised learning, the training data set only consists of example input values and no desired output values are provided. It is therefore up to the machine learning algorithm used to find features by which chunks of data in the data set can be grouped together [14].

2.3 NILM

NILM or Non-Intrusive Load Monitoring is an approach to load monitoring where data from a single energy monitor are disaggregated to determine the appliance-level loads in an electric network [5]. Several different approaches have been studied to achieve this since Hart (1985) [15] first described a prototype non-intrusive appliance load monitor. This section aims to describe the components of a typical NILM algorithm, present an overview of the different ways NILM approaches are classified, and finally, provide an overview of the methods used in evaluating the performance of energy disaggregation algorithms.

Makonin [16] defines 4 steps involved in NILM Algorithms: load monitoring, event detection, feature extraction, and classification. The following subsections explain them in details. Figure 1 visualizes the steps involved in a typical NILM:

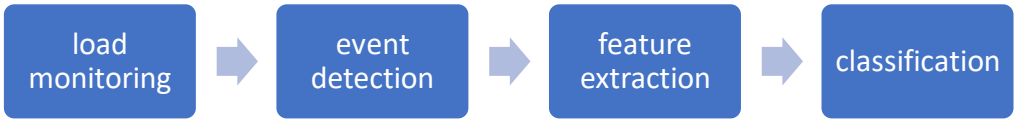

Figure 1. Block diagram of a typical NILM algorithm as defined by Makonin [16]

2.3.1 Event Detection

As described by Makonin [16], the first step to identifying appliances from power measurements is to detect appliance-specific state transition events, such as an appliance being switched on or off. This is achieved by analyzing changes in power consumption. Strategies such as edge detection and probabilistic models have been employed to tackle the issue of event

detection. Probabilistic approaches have less complexity compared to edge detection algorithms and they can be used on data sampled at a lower rate.

2.3.2 Feature Extraction

The following paragraph is based on Makonin [16]. When an event is detected, different features of the monitored signal must be analyzed. There are two types of power signatures, from which features are extracted: transient signatures and steady-state signatures. All appliances are made up of different components, which cause changes in their power signal and create a unique appliance-specific transient signature, which can be used to identify the appliance. In order to capture unique transient signals, electricity signals must be sampled at a high rate. Unlike transient signatures, steady-state signatures do not change rapidly, if at all. This can make it difficult to detect specific appliances because multiple appliances can have similar signatures. Furthermore, the signals of low-powered steady-state appliances can be interpreted as signal noise, which leads to classification errors.

2.3.3 Classification

After the feature extraction stage, the extracted features are processed by machine learning algorithms to identify individual appliances [16]. These algorithms can be classified as supervised learning algorithms, which require appliance-level power consumption data in addition to the total power consumption data for training, and unsupervised learning algorithms, which only require the latter [17].

Supervised methods can be categorized into optimization approaches and pattern recognition approaches [17]. Optimization approaches compare the extracted features against a number of previously stored features to determine a combination of appliances whose total power consumption is closest to the provided aggregate consumption [17]. Pattern recognition approaches have included the use of artificial neural networks [18] and machine learning algorithms such as the hidden Markov model (HMM) [17]. HMM as well as different variations of factorial hidden Markov models (FHMM) used in NILM research are covered in detail in [16]. Unsupervised methods only have access to the aggregate consumption data. Approaches like data mining [19] as well as more complex variations of FHMM [20] have been used to tackle unsupervised NILM.

2.4 NILM Datasets and Evaluation Metrics

Supervised learning algorithms work by analyzing a set of labeled input data and finding characteristics that are later used to label previously unseen data. The load disaggregation

algorithms compared in this thesis are no different: A model is trained on a dataset consisting of real-world time-series energy consumption data and that model is then used to extract the operating states and real-time energy consumption of different appliances from the aggregate consumption data. Therefore, a good dataset is essential. For example, if the training dataset contains no power consumption data for a given appliance, then it will be difficult for the model to accurately predict the power consumption of that appliance. Energy disaggregation datasets generally consist of two types of power measurements [21]:

- Aggregate consumption data – measurements taken from the mains, the total power consumption of the building or system being monitored. Datasets that only contain aggregate consumption data can only be used with unsupervised learning algorithms.
- Appliance or plug-level consumption data. These are the values load disaggregation models aim to predict, given an aggregate value.

In order to objectively compare the accuracy of different disaggregation algorithms, common benchmark datasets are required to highlight the different characteristics of each algorithm. Constructing such datasets can be expensive and time consuming. To overcome this, several open-access datasets of real-world appliance-level household power consumption data have been published to aid in the development of new load disaggregation algorithms [22]. As event-based and event-less NILM approaches require data sampled at different frequencies, disaggregation datasets can be categorized as event-less datasets consisting of low-frequency data and event-based datasets consisting of high-frequency data [21].

2.4.1.1 Low Frequency Datasets

A 2018 review by Pereira & Nunes [21] lists 21 publicly available low-frequency NILM datasets. Of the 21, 17 contain both aggregate and appliance or circuit-evel consumption data. The remaining datasets consist solely of appliance-level consumption data, making it impossible for them to be used as training and testing data simultaneously [21].

2.4.1.2 High Frequency Datasets

Compared to low-frequency datasets, the availability of public high-frequency disaggregation datasets that consist of power data sampled at a rate of 1 kHz or higher is scarce. As of May 5th 2019, at least six real-world high frequency energy disaggregation datasets are publicly available: REDD [23], BLUED [24], UK-DALE [25], PLAID [26], WHITED [27] and HFED [28].

2.4.2 Performance Evaluation

In [29] Makonin presented a unified approach for evaluating the performance of NILM algorithms. Following their evaluation method, there are two types of accuracy that need to be taken into account when evaluating the accuracy of an NILM algorithm: classification accuracy (was the state of the appliance [ON or OFF] predicted correctly?), and estimation accuracy (how accurately was the power consumption of the appliance predicted?). Finally, overall and appliance-level scores of both types of accuracy must be reported.

2.5 NILMTK: A Public NILM Toolkit

NILMTK is an open-source toolkit that was designed to allow researchers to reproducibly analyze load disaggregation algorithms and datasets [30]. Such a platform is important because accurate comparisons can't be made between algorithms that were tested using different datasets and accuracy metrics. NILMTK includes parsers for multiple public real-world power consumption datasets – some of which are mentioned in section 2.4, preprocessing algorithms, statistics for describing datasets, accuracy metrics and implementations of four reference disaggregation algorithms [31].

2.4.1 NILM Algorithms

As of May 5th 2019, the following algorithms are implemented in NILMTK [32]:

- Combinatorial optimization (CO).
- Factorial hidden Markov model (FHMM).
- George Hart's 1985 disaggregation algorithm [15].
- Maximum likelihood estimation.

Several third-party projects have also been made compatible with NILMTK [32]:

- Neural NILM [33] is a project based on the Neural NILM Prototype software [34], which was used in Kelly & Knottenbelt (2015) [18], where the performance of deep neural networks was compared against FHMM and CO using NILMTK.
- Latent Bayesian melding for NILM [35] is a project by Mingjun Zhong, which implements the algorithm discussed in Zhong et al. (2015) [36].
- Neural Disaggregator [37] is a project by Odysseas Krystalakos that contains implementations of five disaggregation algorithms that use artificial neural networks.

2.4.2 Dataset Structure

NILMTK stores data in an HDF5-based open file format. HDF5 is a binary file format that allows hierarchical storage of data. The HDF5 data model consists of the following elements [38,39]:

- Datasets – immutable array-like collections of data. A NILMTK example of this would be power measurements.
- Groups – containers used to store other groups and datasets. For example, every building in NILMTK is a group that contains a group of MeterGroup objects, which are in turn groups and contain a group of ElecMeter objects, which each hold the power consumption data sampled by that meter.
- Attributes – metadata that can be attached to groups and datasets.

In addition to time-series power consumption data, NILMTK also uses metadata to describe each dataset as a whole, as well as buildings and appliances individually [40].

Not all energy disaggregation datasets are created with NILMTK in mind. For example, comma-separated files are used to store data in the REDD dataset To overcome this, NILMTK includes dataset converters capable of converting data to the HDF5 format supported by NILMTK. As of May 5th 2019, NILMTK comes with converters for 11 public energy disaggregation datasets [41].

2.6 OpenHAB

OpenHAB or the open Home Automation Bus is an open-source home automation platform. It allows users to interact with devices connected to their smart home network, schedule automated tasks and monitor the states of their devices via customizable web, Android and iOS interfaces. Although there is no single communications protocol supported by all manufacturers of home automation devices, openHAB supports a large number of devices by many different manufacturers. OpenHAB relies on its community of volunteers who have the ability to add support for new devices to remain vendor-independent. [42]

2.6.1 Architecture and Concepts

OpenHAB's physical layer is comprised of Things [43]. Physical devices, web services and other information sources managed by openHAB are represented as Things [43]. Each Thing provides access to its functionalities through Channels [43]. For instance, a smart energy meter can have a live consumption Channel used for live power readings and another Channel for the total amount of energy consumed since the meter was powered on.

Serving as a link between the physical and the virtual layer are Bindings [42]. Bindings are user-installed openHAB add-ons that handle communications between openHAB Things and their corresponding physical devices [42]. For example, to use a Z-Wave smart meter with openHAB, a Z-Wave Binding is required.

On the virtual layer, Channels can be linked to Items, which represent a Thing's capabilities. When a Channel is linked to an Item, the functions represented by the Item are accessible through that Channel. Items can be used to display information received from devices as well as send commands to devices [44].

However, Items do not necessarily have to be tied to physical devices. An Item can also receive its value from web services, which makes it possible to use Items to represent the live power consumption of individual appliances connected to the local electric network – even unmetered devices whose power consumption is calculated from the aggregate power consumption of the system.

2.6.2 REST API

OpenHAB provides a REST API based on the HTTP protocol that can be used to retrieve data at the Item, Thing and Binding level [45]. This makes it easy for external applications to interact with openHAB. Additionally, the API can be used to change Item states and for certain resources, on-change push notifications are supported [45].

3 Approach

This section presents the architecture and the implementation details of the real-time energy disaggregation solution that was built as part of this thesis. First, the architecture of the real-time NILM system is described. Next, the implementation details are covered, and finally, an overview of the steps taken to bring the system online is presented.

3.1 System Architecture

This section aims oto give an overview of the architecture of the real-time energy disaggregation solution that uses NILMTK for disaggregation and openHAB to visualize data and enable further integration with external applications. Figure 2 depicts the architecture of a real-time NILM solution using openHAB and NILMTK.

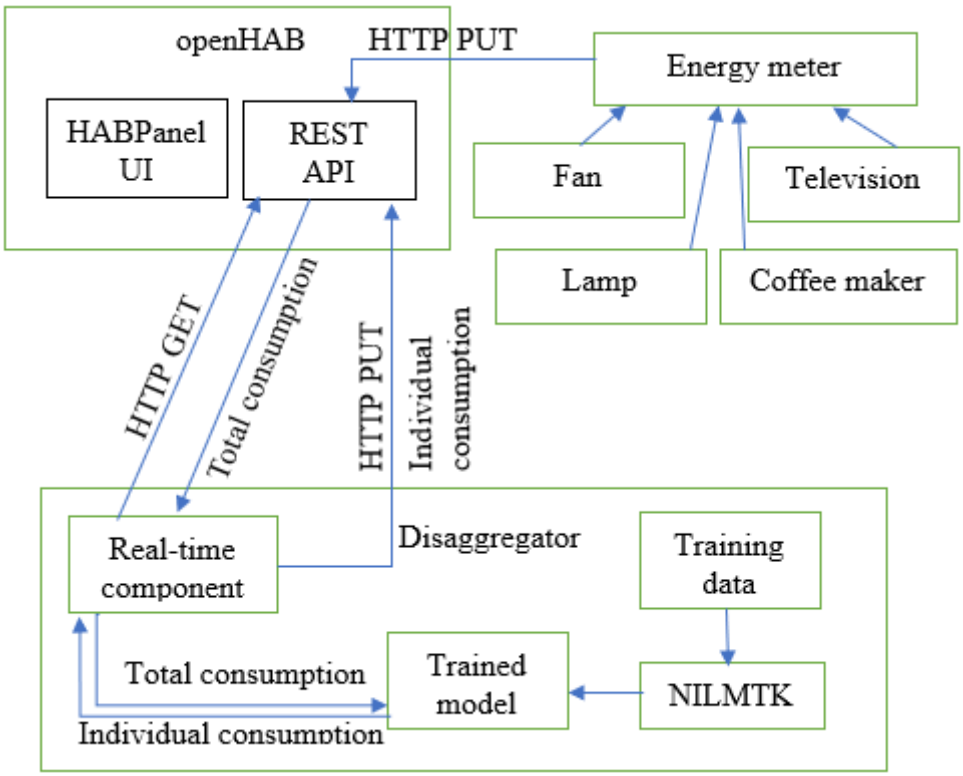

Figure 2. Architecture of a real-time NILM solution using openHAB and NILMTK

For live energy disaggregation to take place, there must also be a live stream of aggregate energy consumption data to be disaggregated. The proposed solution uses a current transformer in conjunction with an Arduino board for live power readings, however, any smart meter that provides live power readings and can be integrated with openHAB can be used to achieve this.

The next component in this design is the disaggregator, which is based on NILMTK. The supervised machine learning algorithms included in NILMTK require training data sets

consisting of time series power consumption data for each appliance to train a model. There are several public data sets of real-world consumption data available online, and some of these are covered in section 2.4. For this thesis, a new data set was created to achieve better accuracy. Once a model is trained, that model can then be used to predict appliance-level loads from the aggregate load. OpenHAB's REST API can be used to retrieve the aggregate consumption data and send the detected individual loads to openHAB, where, in addition to displaying the results to the user via the HABPanel user interface, they are also made easily accessible to external applications via openHAB's APIs and Bindings.

3.2 Implementation Details

The implementation of the presented real-time NILM solution consists of a current transformer, an Arduino board, an openHAB instance running on the local area network, and a laptop running a number of Python scripts that enable communications between the aforementioned components as well as live energy disaggregation. The current transformer is used to measure current in the local electric network. It is interfaced with an Arduino board using EmonLib [46] – a library by OpenEnergyMonitor – to capture live current readings, which are converted to apparent power. The power readings are then transmitted to the laptop over a serial interface. Upon receiving a new reading, the reading is forwarded to a Python script (see appendix 2), which updates the openHAB Item holding the aggregate consumption value using the openHAB REST API covered in section 2.6.2. At this point, the new measurement is visible to users via openHAB's HABPanel graphical interface and accessible to external applications via the REST API.

Stored on the laptop is a data set consisting of appliance-level time series power consumption data. These data are loaded in by a Python script (see appendix 3), which uses NILMTK to train a model based on the provided data. Now, the model can be used to break down aggregate consumption measurements into individual appliance-level loads. The script fetches new aggregate consumption readings from openHAB using the item manager script at a rate of once per second. Every time a new reading is received, the trained model is used to calculate new appliance-level loads. Next, the item manager script (see appendix 2) is used to update the states of all openHAB items with their corresponding appliance-level power consumption as determined by the model. Once the openHAB Items have been updated, the estimated application-level loads are accessible on openHAB.

3.2.1 Measuring Current Consumption

A good real-world dataset is an essential part of any NILM system, since it plays a major role in how accurate the model used for disaggregation will be. It is therefore important that accurate measurements collected from real-world scenarios be used in the training data. The YHDC SCT-013-000 current transformer was used to record consumption data used in the presented real-time NILM solution. It was chosen because of its current rating of 100 A, and its compatibility with Arduino boards using OpenEnergyMonitor's EmonLib library [46,47].

The following circuit, which converts the current signal of the current transformer to a voltage signal with a burden resistor was built by Jakob Mass – the supervisor of this paper – to interface the current transformer with an Arduino:

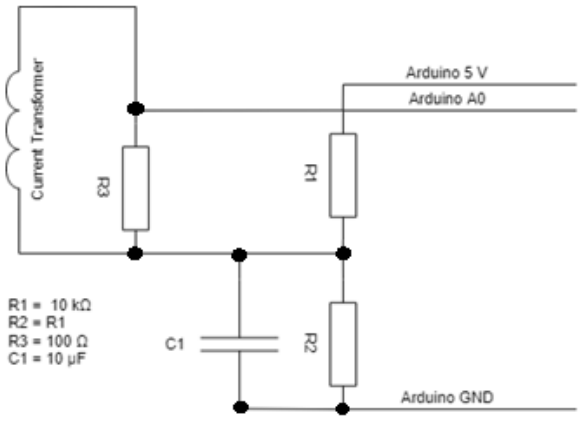

Figure 3. The circuit used to interface the current transformer with an Arduino [47]

3.2.2 Data Conversion

Once the raw data have been acquired for all appliances to be used in training the disaggregation model, the data must be converted to the HDF5 file format supported by NILMTK (see section 2.5.2). In addition to the power measurements, NILMTK also requires that all data sets provide metadata describing the data set in general as well as at the building and appliance level. Knottenbelt & Kelly (2014) [48] provide a detailed overview of The NILM metadata schema used in NILMTK.

To convert the data to the correct format, NILMTK provides a number of converters for existing public datasets (see section 2.5.2). If the power consumption data and metadata are provided in the correct format for the converter, then these dataset converters can be reused to convert new datasets to the HDF5 format (more on this in section 3.3). A cleaner alternative would be to write a new dataset converter tailored to the new data set.

3.2.3 Load Disaggregation

NILMTK was used for load disaggregation. Although the algorithms provided by NILMTK are meant to be used as benchmark algorithms against new disaggregation algorithms, the creation of such an algorithm is not within the scope of this thesis. In the solution presented in this thesis, the disaggregation algorithms included in NILMTK by default are used in training and comparing disaggregation models. NILMTK was chosen as the disaggregation framework for this thesis, due to it being the largest energy disaggregation toolkit on software development platform GitHub [49], the only disaggregation toolkit that is, as of May 7th, 2019, in active development, and it has been used in numerous related works [33,50,51]. NILM-Eval [52] is an alternative to NILMTK that is covered in more detail and compared to NILMTK in Beckel et al. (2014) [53]. However, as of May 7th, 2019, the code of that project has not been updated in nearly four years.

3.2.4 OpenHAB

OpenHAB is used to visualize both the aggregate as well as the appliance-level power consumption data. OpenHAB was chosen as the home automation platform for this thesis because it is open-source, has a large community, thorough documentation, and support for integration with external applications, see section 2.6 for more. The use of openHAB as a middleware between the power meter providing live readings and the disaggregator using said readings means the solution is not dependent on a specific type of meter: Different channels capable of communicating with openHAB can be used as information sources for live power readings, even web services. Alternatives to openHAB include Home Assistant [54], ioBroker [55], and PiDome [56].

3.3 Activities

The following section gives an overview of the steps that were taken to make the real-time disaggregator work. The creation of the data set is detailed and the process of configuring openHAB to work with appliances from the data set is described.

3.3.1 Creating the Dataset

A data set consisting of a series of power measurements of 4 appliances was created. The following appliances were recorded: a 40 W light bulb, a television, an electric air heater, and a coffee maker. The first 3 were chosen because their power consumption does not fluctuate much, which means they should be easy to detect. The coffee maker was chosen because it was expected that the constantly changing power demand as it went through different stages (e.g.

making the coffee, washing), would make it harder for NILM algorithms to accurately predict the operating states of both the coffee maker as well as other appliances.

The data set was recorded at the Internet of Things and Smart Solutions lab. Due to time constraints, each appliance was only recorded for an hour. An exception is the coffee maker, which was recorded for 10 minutes. As a result, the data had to be artificially replicated to make testing possible. Additionally, as only a single energy monitor was used in making the data set, testing data were generated by adding up the individual loads at all time points in the data set. This is also suboptimal, since no noise caused by other devices connected to the electric network is recorded, which is rarely the case in real world scenarios.

Live power readings are provided by the current sensor covered in section 3.2.1 at a rate of 3 Hz. An Arduino script (depicted in figure 4) handles the live inputs from the sensor on an Arduino board.

```
// EmonLibrary examples openenergymonitor.org, Licence GNU GPL V3
// For Serial Plotter
#include "EmonLib.h"                   // Include Emon Library
#include "Timer.h"

EnergyMonitor emon1;                   // Create an instance
Timer t;

void setup()
{
  Serial.begin(9600);
  // e.g. calibration  - CT Ratio / Burden resistance = (100A / 0.05A) / 33 Ohms = 60.6

  //emon1.current(A0, 60.5f);             // Current: input pin, calibration.

  // set burden resistance = 101 Ω, 2000/101=19.8

  emon1.current(A0, calibration);

}


void loop()
{
  double Irms = emon1.calcIrms(1480);  // Calculate RMS current (1480: no. of samples)
  Irms = max(0.0f, Irms - 0.03f); // a little offset to 'fake' a nice near 0.0 value idle reading

  double pwr = Irms*238.0f; //Apparent power

  Serial.print(pwr,4);
  Serial.print("\n");
}
```

Figure 4. Arduino script in charge of communicating with the CT sensor

The live current readings are converted to apparent power, assuming that the voltage has a constant value of 238 V, which was measured at the test environment from a wall outlet with a multimeter. The setup used in creating the dataset can be seen in figure 5.

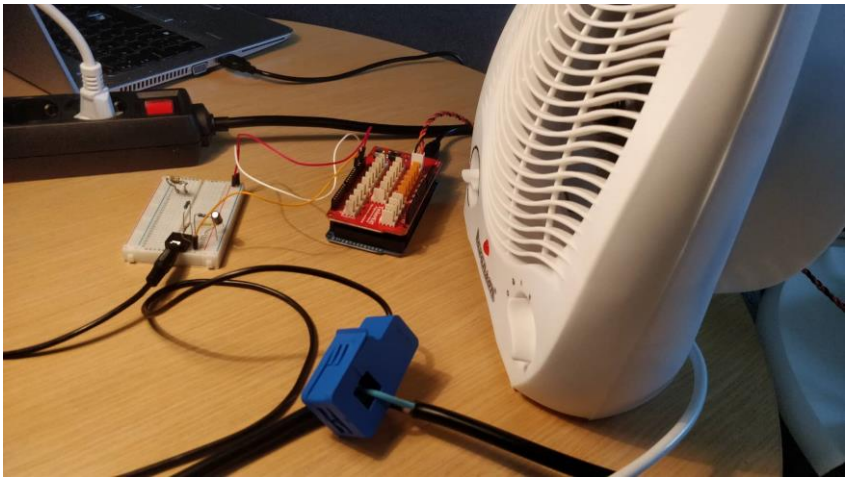

Figure 5. Power consumption of the electric air heater being measured

The data are transmitted to a Windows laptop over USB, where they are read by a Python script (see appendix 4), which uses pySerial [57] for serial communications. Each appliance has its own dedicated text file of power consumption data. Every time a new reading is received, a line consisting of a UNIX timestamp – time elapsed since midnight January 1st, 1970 (UTC) in seconds – and the power reading, separated by a comma, are written into the file.

After recording and saving the initial power consumption data for all appliances, another Python script (see appendix 5) is used to further manipulate the data. Since NILMTK does not support sampling rates higher than 1 Hz for low-frequency NILM and the recorded power data were sampled at a higher frequency, the timestamps are modified to be a second apart in each file. This is not ideal, since the power signatures of the appliances in the training and testing data are different compared to power readings received from openHAB in real time, since those power readings are not manipulated in this way. Next, the appliance-level data are saved in the same format as the data in the REDD dataset: text files, where each row consists of a UNIX timestamp, a whitespace character and a power consumption value.

As the data are saved in the same format as the data in the REDD dataset (see section 2.4.1.1), the REDD dataset converter included in NILMTK was used to convert the data to the format supported by NILMTK. For this, some changes had to be made in NILMTK metadata files concerning the built-in REDD dataset converter (see appendix 6), after which, the data could be converted using the REDD dataset converter and loaded into NILMTK to be used as training and testing data.

3.3.2 Configuring openHAB and Disaggregation

The next step in setting up the real-time disaggregator is to create an openHAB Item (see section 2.6.1) for each appliance. The name of each openHAB Item must match the label assigned to the appliance in the metadata. OpenHAB's HABPanel interface [58] can then be configured to show the values of the newly created items (figure 6).

Figure 6. User interface displaying real-time power consumption information

Next, the Arduino setup mentioned in section 3.3.1 can be used in conjunction with a Python script (see appendix 1) to read the total consumption of all appliances connected to the local electric network and update the aggregate load value in openHAB.

Once the live load monitoring system is in place, the disaggregator script (see appendix 3) can be started. First, a model is trained using a training dataset. Next, aggregate power consumption values are read from openHAB in real time using the REST API. Every time an aggregate consumption value is received from openHAB, it is given to the trained model as input. The resulting appliance-level loads are further filtered by using previously defined minimum power-on threshold values for each appliance in NILMTK metadata and ignoring results, where a power consumption value greater than zero but lower than the minimum power-on threshold value for that appliance was received from the model to help reduce classification errors. Such classification errors can occur as a result of noise in the training data. All openHAB Items are updated with the new appliance-level power consumption values over the REST API and a new aggregate consumption value is fetched to start the disaggregation process again.

4 Evaluation

The aim of this chapter is to evaluate the accuracy of the presented real-time NILM solution. Evaluations are done in the context of the data set described in section 3.3.1. NILMTK implementations of two algorithms – a combinatorial optimization algorithm and FHMM – are

tested. Two aspects are compared: The classification accuracy and the estimation accuracy. All evaluations are done using a dedicated Python script (see appendix 7).

4.1 Accuracy Metrics

To evaluate the classification accuracy of the disaggregator models, F-score is used. It is defined in [29] as the harmonic mean of precision and recall, where precision represents the ratio of correctly predicted positive values and all predicted positive values, and recall is the number of correctly predicted positive values over all positive values:

$$F_1 = 2 \times \frac{precision \times recall}{precision + recall},\ precision = \frac{tp}{tp + fp},\ recall = \frac{tp}{tp + fn},$$

where *tp* represents true-positives, meaning it was correctly predicted that the appliance was on, *fp* represents false-positives, meaning it was incorrectly predicted that the appliance was on, and *fn* represents false-negatives, meaning it was incorrectly predicted that the appliance was off. An F-score value of 1 indicates perfect accuracy, and 0, the lowest possible accuracy.

To evaluate the estimation accuracy of the models, root-mean-squared error (RMSE) is used – the lower the result, the closer the predicted power consumption value is to the actual consumption value.

4.2 Data set

A data set consisting of power measurements of four appliances – a television, an electric air heater, a 40W light bulb, and a coffee maker – is used for evaluation. The data set consists of 3 hours of individual load consumption data for each appliance. The making of this data set is described in detail in section 3.3.1. The data set – visualized in figure 7 – features recordings of all possible operating states of the appliances whose data are used in training the disaggregation models.

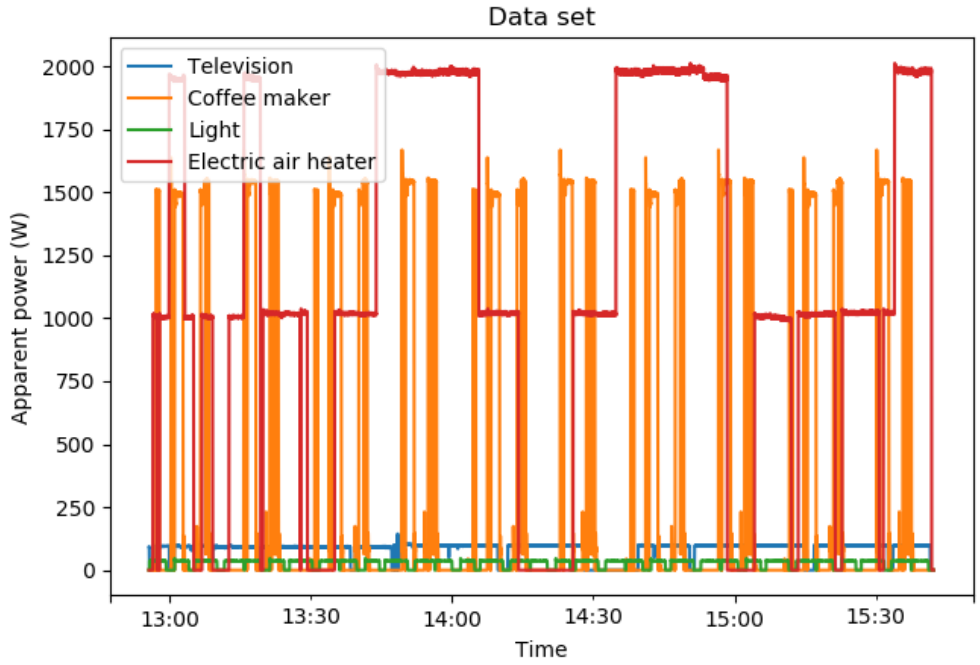

Figure 7. Visualization of the data set

In all tests, the data set was split into two: A training data set, which consists of the first 90 percent of the data, and a test data set, which consists of the remaining 10%. The idea behind this is that a model should not be evaluated on the same data that were used in training to avoid overly optimistic results. An exception to this is the coffee maker, which was only recorded for a short period of time and the consumption data were artificially replicated to fill the 3 hour time period. Therefore, the recordings concerning the coffee maker are identical in both data sets. Figures 8 and 9 depict the training data set and the test data set, respectively.

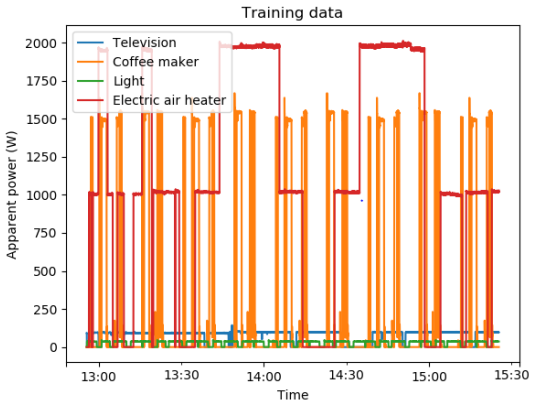

Figure 8. Training data

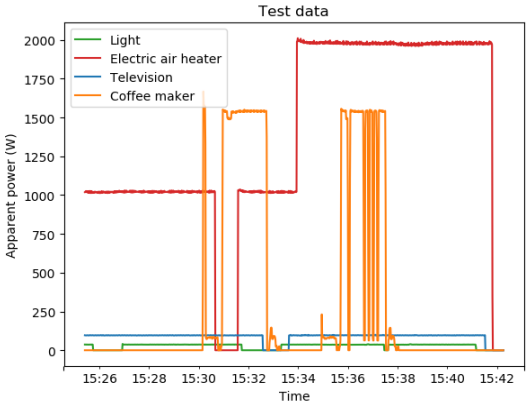

Figure 9. Test data

4.3 Factorial Hidden Markov Model Accuracy

In this section, the accuracy of disaggregation models trained using the FHMM algorithm is evaluated. The accuracy of the algorithm is evaluated in two scenarios: with the coffee maker and without the coffee maker. This is done because the data recorded for the coffee maker are of lower quality than the data recorded for other appliances, since it was recorded for a shorter period of time than other appliances in the data set and the data were artificially replicated to fill 3 hours. Additionally, the coffee maker has more operating states compared to the other appliances tested. Figure 10 visualizes the ground truth or expected power consumption values, while figures 11 and 12 depict the predictions of models trained without the coffee maker and with the coffee maker, respectively.

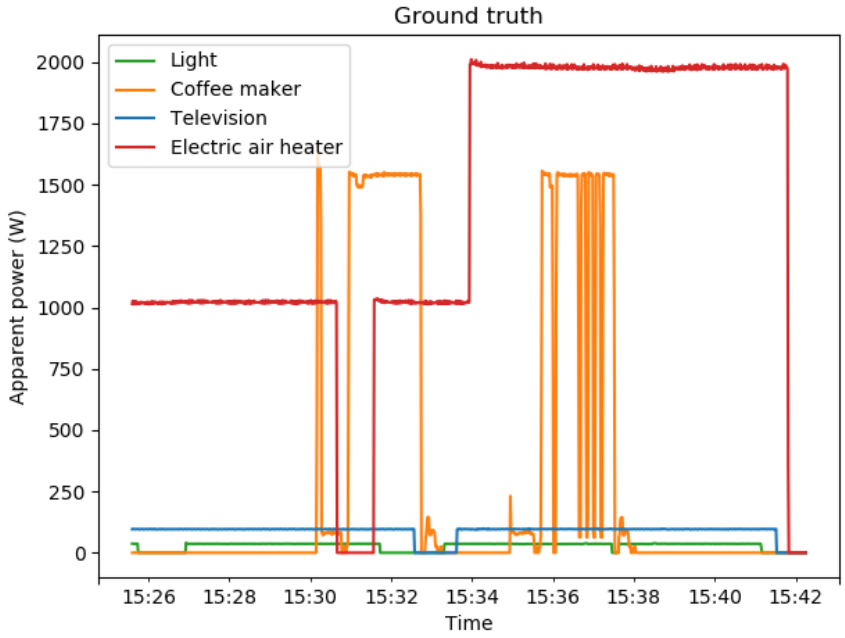

Figure 10. Actual power consumption at the appliance level

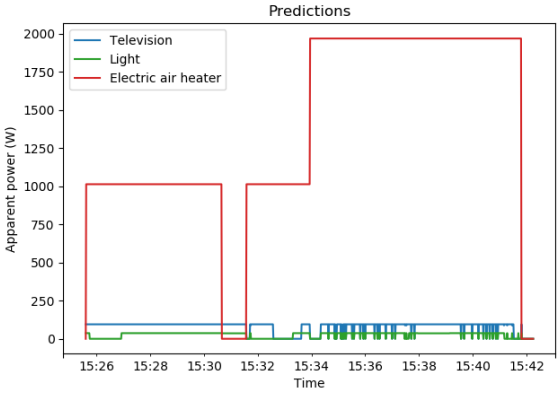

Figure 11. Predictions without the coffee maker

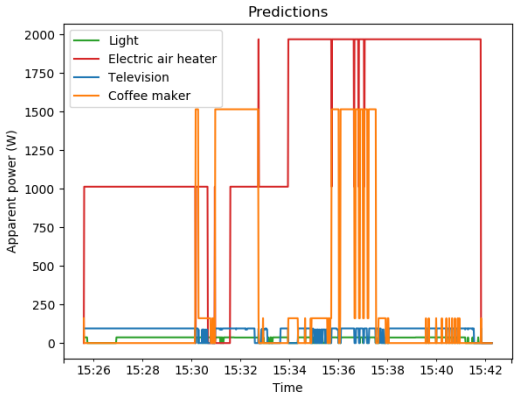

Figure 12. Predictions with the coffee maker

In both scenarios, the models managed to accurately predict the operating states of the electric air heater, though the addition of the coffee maker did have a slight negative effect on the accuracy. The television and the 40 W light bulb were classified correctly until the heat was turned up on the electric air heater and the power consumption doubled. After this, the models struggled to differentiate between the television and the light bulb. It is possible that this was caused by the relatively small data set that was used and that accuracy could be improved by gathering more training data. Classification and estimation accuracy results are presented in table 1.

Appliance	**F-Score** without coffee maker	**F-score** with coffee maker	**RMSE** without coffee maker (W)	**RMSE** with coffee maker (W)
Electric air heater	0.999	0.998	35.139	88.9
Light	0.949	0.879	9.953	15.456
Television	0.961	0.919	24.961	35.503
Coffee maker	-	0.907	-	95.019
Mean	0.97	0.926	23.351	58.72

Table 1. FHMM accuracy

Classification accuracy scores are high: a mean of 0.97 across all appliances without the coffee maker and 0.926 with the coffee maker. The electric air heater was almost always classified correctly in both scenarios: an F-score of 0.999 without the coffee maker and 0.998 with the

coffee maker. Classification accuracy of the 40 W light bulb suffered the most with the addition of the coffee maker: a drop of 7.4 %.

Estimation accuracy is high for the electric air heater, but low for the low-powered appliances. Since the electric air heater has a load demand of roughly 1 kW or 2 kW depending on the operating state, an RMSE score of 35.139 W means an error of 3.5% or less. For a 40 W bulb, however, an RMSE score of 9.953 W is a significant miss at 25%. The addition of the coffee maker had a severe negative impact on the estimation accuracy for all appliances: the RMSE values increased by at least 29% across the board.

4.4 Combinatorial Optimization Accuracy

In this section, the accuracy of the disaggregation models trained using the NILMTK implementation of a combinatorial optimization algorithm is evaluated. The accuracy of the algorithm is again evaluated in two scenarios: first on a model trained with the coffee maker and then on a model trained without the coffee maker for the reasons mentioned in section 4.3. Figure 13 visualizes the ground truth or expected power consumption values, while figures 14 and 15 depict the predictions of models trained without the coffee maker and with the coffee maker, respectively.

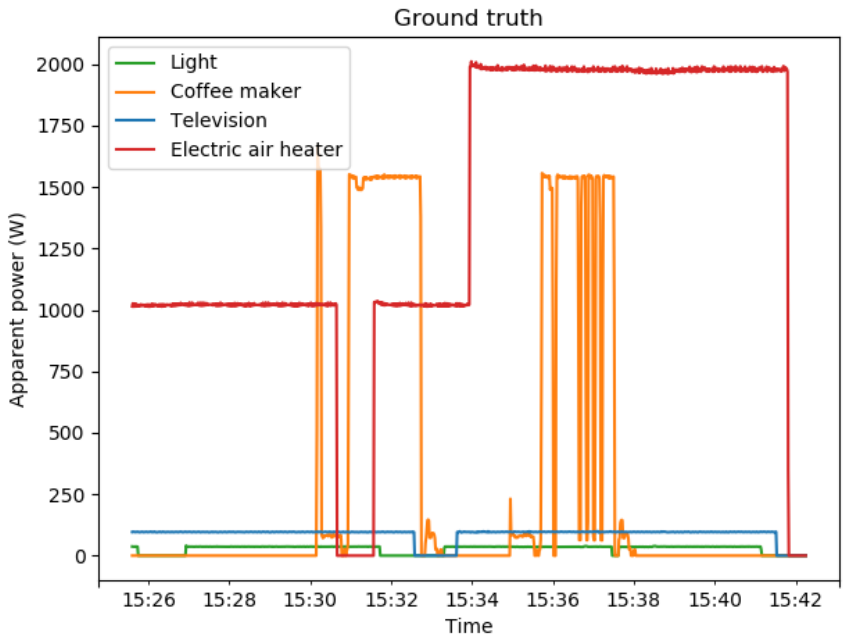

Figure 13. Actual power consumption at the appliance level

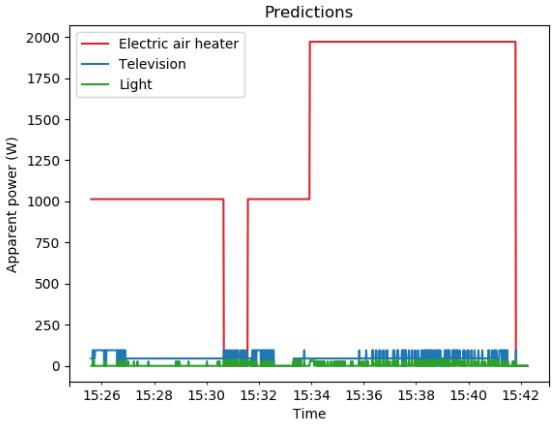

Figure 14. Predictions without the coffee maker

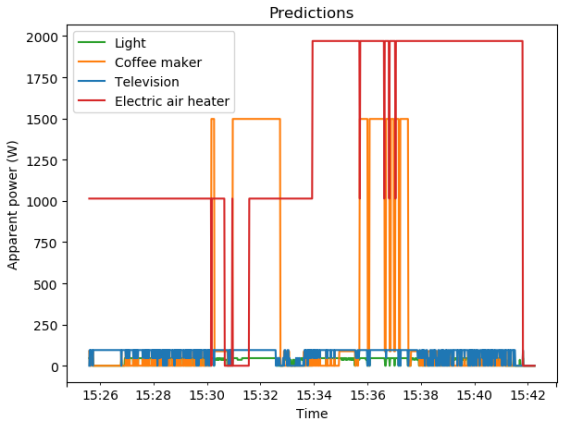

Figure 15. Predictions with the coffee maker

As can be seen from figures 12 and 13, the combinatorial optimization algorithm also manages to accurately determine the operating states of the electric air heater, though, similarly to the FHMM results, the introduction of the coffee maker does have a slight negative impact on accuracy. The model struggles to classify the television and the 40 W light bulb even when the heater is not turned all the way up. Classification and estimation accuracy results are presented in table 2.

Appliance	**F-Score** without coffee maker	**F-score** with coffee maker	**RMSE** without coffee maker (W)	**RMSE** with coffee maker (W)
Electric air heater	0.999	0.998	14.131	77.236
Light	0.391	0.867	28.051	18.848
Television	0.963	0.943	46.755	33.132
Coffee maker	-	0.760	-	83.649
Mean	0.784	0.892	29.646	53.216

Table 2. CO accuracy

The classification accuracy of the electric air heater and the television are high, with the F-scores staying above 0.94 in spite of the coffee maker. The same can't be said for the 40 W bulb, however: an F-score of 0.391 without the coffee maker and 0.867 with the coffee maker. This is interesting because in all other cases, the inclusion of the coffee maker in the data set

has had a negative impact on the classification accuracy of other appliances. The classification accuracy of the coffee maker is also fairly low: an F-score of 0.76.

The estimation accuracy of the electric air heater is once again high, with an error of 14.131 W without the coffee maker and 77.236 W with the coffee maker. The estimation accuracy was much lower for the 40 W bulb and the television. With the average load demand of the 40 W bulb being 40 W, the reported RMSE score of 28.051 W translates to an error of 70.1%. RMSE scores improved for both the television as well as the light bulb when the coffee maker was also considered, but worsened for the electric air heater.

4.5 Summary

The FHMM approach consistently outperformed the combinatorial optimization algorithm in both classification as well as load estimation. Overall, the classification results of the FHMM models – average F-scores of 0.926 or higher – mean that such models can be used to accurately determine the operating states of different appliances from a single energy meter's readings. Errors in load estimation were fairly low for the electric air heater. However, the models performed much worse at estimating the load consumption of lower-powered appliances like the television and the 40 W light bulb.

These tests were conducted in a controlled test environment and with quite limited data. On the one hand, this means that the results can be overly optimistic, since no noise in the form of other unmetered appliances was present in the data set. On the other hand, the small data set size means that the models might not have fully learned the operating states of some appliances, and therefore, accuracy was reduced. For a more accurate evaluation of these models, a larger data set consisting of power measurements from real-world scenarios should be used.

5 Conclusions

In this work, an openHAB-compatible real-time NILM solution was proposed. It provides the user access to appliance-level energy consumption data using only a single energy meter. To achieve this, a data set consisting of power consumption data was created. Next, a machine learning model was trained to break down aggregate energy measurements into appliance-level energy consumption data using NILMTK. Finally, an openHAB environment was set up to visualise the data and allow further integration with other applications. Since openHAB was used as the source for real-time aggregate power measurements, no modifications are required to use the real-time disaggregator with a different energy monitor.

The solution was tested with 4 appliances – a 40 W light bulb, a television, an electric air heater, and a coffee maker. It was found that both disaggregation algorithms used in testing – FHMM and a combinatorial optimization algorithm – are capable of recognizing larger loads, however, struggle to identify appliances whose individual loads only make up a small portion of the total power consumption. The small data set used in these tests likely plays a role in this.

The electric air heater's operating state was almost always predicted correctly. This is evidenced by its nearly perfect F-score of 0.998 – 0.999 across all test scenarios. The light bulb, however, was often confused with the television when larger loads were introduced, resulting in an F-score of 0.391 in one scenario. The FHMM approach consistently outperformed the combinatorial optimization algorithm, achieiving average F-scores higher than 0.92 in both test scenarios. However, both algorithms struggled to accurately estimate the load consumption of the television and the 40 W light bulb. These results mean that in practice, while this approach can be counted on to determine which appliances are operating, it can't be relied on to accurately estimate the power consumption of the detected appliances – especially those whose power consumption only makes up an insignificant amount of the total power consumption.

5.1 Future Work

A shortcoming of the practical portion of this work is the data set: it contains too few samples of power measurements to properly train models and it is artificially modified. Additionally, since power measurements were taken over a short period of time and in a dedicated test environment as opposed to someone's home, additional information like correlation of usage with other appliances cannot be leveraged. Therefore, a larger data set should be created to

improve accuracy, but also enable the use of disaggregation algorithms making use of neural networks. Furthermore, unsupervised approaches should be investigated, since no appliance-level power measurements are required when training models, which makes setup much easier.

6 References

1. Gubbi J, Buyya R, Marusic S, Palaniswami M. Internet of Things (IoT): A vision, architectural elements, and future directions. Future generation computer systems. 2013 Sep 1;29(7):1645-60.
2. 2019. Global Energy & CO2 Status Report. https://www.iea.org/geco/electricity/ [cited May 10 2019]
3. Smart metering implementation programme: Progress Report for 2018 [Internet]. Department for Business, Energy & Industrial Strategy; 2018 [cited May 10 2019]. Available from: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/767128/smart-meter-progress-report-2018.pdf
4. Ehrhardt-Martinez et al. Advanced Metering Initiatives and Residential Feedback Programs: A Meta-Review for Household Electricity-Saving Opportunities [Internet]. American Council for an Energy-Efficient Economy; 2010 [cited May 10 2019]. Available from: https://www.smartgrid.gov/files/ami_initiatives_aceee.pdf
5. Herrero JR, Murciego ÁL, Barriuso AL, de La Iglesia DH, González GV, Rodríguez JM, Carreira R. Non intrusive load monitoring (NILM): A state of the art. In International Conference on Practical Applications of Agents and Multi-Agent Systems 2017 Jun 21 (pp. 125-138). Springer, Cham
6. Kelly J, Knottenbelt W. Neural NILM: Deep neural networks applied to energy disaggregation. In Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments 2015 Nov 4 (pp. 55-64). ACM
7. Dinesh C, Nettasinghe BW, Godaliyadda RI, Ekanayake MP, Ekanayake J, Wijayakulasooriya JV. Residential appliance identification based on spectral information of low frequency smart meter measurements. IEEE Transactions on smart grid. 2016 Nov;7(6):2781-92.
8. Zeifman M, Roth K. Nonintrusive appliance load monitoring: Review and outlook. IEEE transactions on Consumer Electronics. 2011 Feb;57(1):76-84
9. An Introduction to AC Power [Internet]. OpenEnergyMonitor. [cited May 10 2019]. Available from: https://learn.openenergymonitor.org/electricity-monitoring/ac-power-theory/introduction
10. Understanding the Zero Carbon Energy System [Internet]. OpenEnergyMonitor. 2017 [cited May 10 2019]. Available from: https://learn.openenergymonitor.org/sustainable-energy/energy/introduction
11. How to Build an Arduino Energy Monitor – Measuring Mains Current Only [Internet]. OpenEnergyMonitor. [cited May 10 2019]. https://learn.openenergymonitor.org/electricity-monitoring/ct-sensors/how-to-build-an-arduino-energy-monitor-measuring-current-only
12. Emoncms [Internet]. Emoncms. [cited May 10 2019]. Available from: https://emoncms.org/

13. Voyant C, Notton G, Kalogirou S, Nivet ML, Paoli C, Motte F, Fouilloy A. Machine learning methods for solar radiation forecasting: A review. Renewable Energy. 2017 May 1;105:569-82.
14. Lison P. An introduction to machine learning [Internet]. 2012 [cited May 10 2019]. Available from: http://folk.uio.no/plison/pdfs/talks/machinelearning.pdf
15. Hart GW. Prototype nonintrusive appliance load monitor. In MIT Energy Laboratory Technical Report, and Electric Power Research Institute Technical Report 1985 Sep.
16. Makonin S. Approaches to non-intrusive load monitoring (nilm) in the home. 2012 Oct 24.
17. Zoha A, Gluhak A, Imran M, Rajasegarar S. Non-intrusive load monitoring approaches for disaggregated energy sensing: A survey. Sensors. 2012 Dec;12(12):16838-66
18. Kelly J, Knottenbelt W. Neural NILM: Deep neural networks applied to energy disaggregation. In Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments 2015 Nov 4 (pp. 55-64). ACM
19. Shao H, Marwah M, Ramakrishnan N. A temporal motif mining approach to unsupervised energy disaggregation: Applications to residential and commercial buildings. In Twenty-Seventh AAAI Conference on Artificial Intelligence 2013 Jun 29.
20. Kim H, Marwah M, Arlitt M, Lyon G, Han J. Unsupervised disaggregation of low frequency power measurements. In Proceedings of the 2011 SIAM international conference on data mining 2011 Apr 28 (pp. 747-758). Society for Industrial and Applied Mathematics.
21. Pereira L, Nunes N. Performance evaluation in non-intrusive load monitoring: Datasets, metrics, and tools—A review. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery. 2018 Nov;8(6):e1265
22. ILM datasets [Internet]. NILM wiki. [cited May 10 2019]. Available: http://wiki.nilm.eu/datasets.html
23. Kolter JZ, Johnson MJ. REDD: A public data set for energy disaggregation research. In Workshop on Data Mining Applications in Sustainability (SIGKDD), San Diego, CA 2011 Aug 21 (Vol. 25, No. Citeseer, pp. 59-62).
24. Anderson K, Ocneanu A, Benitez D, Carlson D, Rowe A, Berges M. BLUED: A fully labeled public dataset for event-based non-intrusive load monitoring research. In Proceedings of the 2nd KDD workshop on data mining applications in sustainability (SustKDD) 2012 Aug 12 (pp. 1-5).
25. Kelly J, Knottenbelt W. The UK-DALE dataset, domestic appliance-level electricity demand and whole-house demand from five UK homes. Scientific data. 2015 Mar 31;2:150007.
26. Gao J, Giri S, Kara EC, Bergés M. Plaid: a public dataset of high-resoultion electrical appliance measurements for load identification research: demo abstract. In proceedings of the 1st ACM Conference on Embedded Systems for Energy-Efficient Buildings 2014 Nov 3 (pp. 198-199). ACM.

27. Kahl M, Haq AU, Kriechbaumer T, Jacobsen HA. Whited-a worldwide household and industry transient energy data set. In 3rd International Workshop on Non-Intrusive Load Monitoring 2016.
28. Gulati M, Ram SS, Singh A. An in depth study into using EMI signatures for appliance identification. In Proceedings of the 1st ACM Conference on Embedded Systems for Energy-efficient Buildings 2014 Nov 3 (pp. 70-79). ACM
29. Makonin S, Popowich F. Nonintrusive load monitoring (NILM) performance evaluation. Energy Efficiency. 2015 Jul 1;8(4):809-14.
30. Batra N, Kelly J, Parson O, Dutta H, Knottenbelt W, Rogers A, Singh A, Srivastava M. NILMTK: an open source toolkit for non-intrusive load monitoring. In Proceedings of the 5th international conference on Future energy systems 2014 Jun 11 (pp. 265-276). ACM.
31. NILMTK: Non-Intrusive Load Monitoring Toolkit. Github [Internet]. [cited May 10 2019]. Available from: https://github.com/nilmtk/nilmtk
32. NILM Algorithms. Github [Internet]. [cited May 10 2019]. Available from: https://github.com/nilmtk/nilmtk/wiki/NILM-Algorithms
33. Neuralnilm. Github [Internet]. [cited May 10 2019]. Available from: https://github.com/JackKelly/neuralnilm
34. Neural NILM Prototype. Github [Internet]. [cited May 10 2019]. Available from: https://github.com/JackKelly/neuralnilm_prototype
35. Latent Bayesian melding for non-intrusive load monitoring (energy disaggregation). Github [Internet]. [cited May 10 2019]. Available from: https://github.com/MingjunZhong/LatentBayesianMelding
36. Zhong M, Goddard N, Sutton C. Latent Bayesian melding for integrating individual and population models. In Advances in neural information processing systems 2015 (pp. 3618-3626).
37. Code for NILM experiments using Neural Networks. Github [Internet]. [cited May 10 2019]. Available from: https://github.com/OdysseasKr/neural-disaggregator
38. Collette A. Python and HDF5: Unlocking Scientific Data. " O'Reilly Media, Inc."; 2013 Oct 21.
39. Datasets. H5py [Internet]. [cited May 10 2019]. Available from: http://docs.h5py.org/en/stable/high/dataset.html
40. NILM METADATA. Github [Internet]. [cited May 10 2019]. Available from: https://github.com/nilmtk/nilm_metadata
41. Dataset converters. Github [Internet]. [cited May 10 2019]. Available from: https://github.com/nilmtk/nilmtk/tree/master/nilmtk/dataset_converters
42. Documentation. OpenHAB [Internet]. [cited May 10 2019]. Available from: https://www.openhab.org/docs/
43. Things. OpenHAB [Internet]. [cited May 10 2019]. Available from: https://www.openhab.org/v2.3/docs/configuration/things.html

44. Items. OpenHAB [Internet]. [cited May 10 2019]. Available from: https://www.openhab.org/docs/configuration/items.html
45. openHAB REST API. OpenHAB [Internet]. [cited May 10 2019]. Available from: https://www.openhab.org/docs/configuration/restdocs.html
46. emonLib. Github [Internet]. [cited May 10 2019]. Available from: https://github.com/openenergymonitor/EmonLib
47. CT Sensors – Interfacting with an Arduino. OpenEnergyMonitor [Internet]. [cited May 10 2019]. Available from: https://learn.openenergymonitor.org/electricity-monitoring/ct-sensors/interface-with-arduino
48. Kelly J, Knottenbelt W. Metadata for energy disaggregation. In 2014 IEEE 38th International Computer Software and Applications Conference Workshops 2014 Jul 21 (pp. 578-583). IEEE.
49. Github. Github [Internet]. [cited May 10 2019]. Available from: https://github.com/
50. Arjunan P, Singh A, Singh P. 2018. Middleware systems and analytics for energy management in buildings (Doctoral dissertation, IIIT-Delhi).
51. Buneeva N, Reinhardt A. AMBAL: Realistic load signature generation for load disaggregation performance evaluation. In 2017 IEEE International Conference on Smart Grid Communications (SmartGridComm) 2017 Oct 23 (pp. 443-448). IEEE.
52. NILM-Eval. Github [Internet]. [cited May 10 2019]. Available from: https://github.com/beckel/nilm-eval
53. Beckel C, Kleiminger W, Cicchetti R, Staake T, Santini S. The ECO data set and the performance of non-intrusive load monitoring algorithms. In Proceedings of the 1st ACM Conference on Embedded Systems for Energy-Efficient Buildings 2014 Nov 3 (pp. 80-89). ACM.
54. Home Assistant. Home Assistant [Internet]. [cited May 10 2019]. Available from: https://www.home-assistant.io/
55. ioBroker. ioBroker [Internet]. [cited May 10 2019]. Available from: http://www.iobroker.net/
56. PiDome - Home Automation. PiDome [Internet]. [cited May 10 2019]. Available from: https://pidome.org/
57. pySerial. Github [Internet]. [cited May 10 2019]. Available from: https://github.com/pyserial/pyserial
58. Designing dashboard interfaces with HABPanel. OpenHAB [Internet]. Available from: https://www.openhab.org/docs/configuration/habpanel.html

7 Appendices

1 Serial Reader

```
import serial
import item_manager

#A script that continuously loads in new power consumption values
#from an Arduino board using pyserial

ser = serial.Serial('COM3', 9600, timeout=0)
ser.flushInput()
previous = ""

updater = item_manager.ItemManager()

while True:
    if ser.in_waiting:
        current = ser.read().decode("utf-8")
        if len(current) == 0:
            continue
        if current == "\n" and len(previous) > 0:   #new power value received
            previous = float(previous)
            updater.update_item("Site meter", previous) #update aggregate consumption on openHAB
            previous = ""
            continue
        #inefficient, however, solutions just looking for line changes would occasionally not work
        previous += current
```

2 OpenHAB Item Manager

```
import requests
import json

#openHAB item manager class - used to interact with openHAB Items
class ItemManager:
    def __init__(self, base_url = "http://localhost:8080"):
        self.base_url = base_url + "/rest/items/"

    #creates an openHAB Item with the provided name and assigns it the appliances group
    def create_item(self, item):
        print("Creating item: " + item)
        body = {
            "type": "Number",
            "name": item.replace(" ", "_"),
            "category": "appliances",
            "state": 0.0000,
            "tags": [],
            "groupNames": ["appliances"],
            "groupType": "appliances"
        }
        requests.put(
            url=self.base_url + item,
            headers={"Content-Type": "application/json"},
            data=json.dumps(body)
        )

    #given an openHAB Item name and value, updates the corresponding openHAB Item value
    def update_item(self, item, value):
        print("Updating item: " + item + ", value: " + str(value))
        item = item.replace(" ", "_")
        requests.put(
            url=self.base_url + item + "/state",
            headers={'Content-type': 'text/plain'},
            data=str(value)
        )

    #returns the latest aggregate consumption reading from openHAB, -1 is returned if unable to get reading
    def get_aggregate(self):
        url = self.base_url + 'Site_meter'
        print(url)
        try:
            req = requests.get(url, params={'type': 'json'}, headers={'Content-type': 'text/plain'})
            return req.json()["state"]
        except:
            print("unable to get aggregate consumption")
            return -1
```

3 Live Disaggregator

The Python script used for real time disaggregation can be found in the included ZIP archive as live_disaggregator.py.

4 Script to Save Power Readings

```
import serial
import time
import datetime

def save_reading(s, path):
    f = open(path, "w")
    f.write(str(time.time()) + "," + s + "\n")
    f.close()

def start_recording(path):
    ser = serial.Serial('COM3', 9600, timeout=0)
    ser.flushInput()
    previous = ""
    i = 0
    while True:
        if ser.in_waiting:
            current = ser.read().decode("utf-8")
            if len(current) == 0:
                continue
            if current == "\n" and len(previous) > 0:
                save_reading(previous, path)
                previous = ""
                i += 1
                print(i)
                continue
            previous += current

filename = "telekas.txt"
start_recording(filename)
```

5 Script to Convert Power Readings to the REDD format

```
def save_file(input_file, output_file, times):
    f = open(input_file, encoding="utf-8")
    lines = f.readlines()
    f.close()
    pwr = [line.strip("\n").split(",")[1] for line in lines[:10000] if line.strip("\n") != ""]
    time_copy = round(float(times[0]))
    f = open(output_file, "w")

    for i in range(10000):
        f.write(str(time_copy) + " " + pwr[i] + "\n")
        time_copy += 1
    f.close()

files = ["lamp.txt", "puhur.txt", "telekas.txt", "kohvimasin.txt"]
pwrs = []
times = []
for i in range(len(files)):
    f = open(files[i], encoding="utf-8")
    lines = f.readlines()
    f.close()
    if len(times) == 0:
        times = [line.strip("\n").split(",")[0] for line in lines[:10000] if line.strip("\n") != ""]
    f.close()
    pwr_first = [float(line.strip("\n").split(",")[1]) for line in lines[:10000] if line.strip("\n") != ""]
    pwrs.append(pwr_first)

f = open("data/redd/low_freq/house_1/channel_1.dat", "w")
time_copy = round(float(times[0]))
start = time_copy - 10000

for i in range(10000):
    pwr = sum([0 if i  >  8663 and power == pwrs[3] else power[i] for power in pwrs])
    f.write(str(time_copy) + " " + str(pwr) + "\n")
    time_copy += 1

f.close()

save_file("lamp.txt", "data/redd/low_freq/house_1/channel_2.dat", times)
save_file("puhur.txt", "data/redd/low_freq/house_1/channel_3.dat", times)
save_file("telekas.txt", "data/redd/low_freq/house_1/channel_4.dat", times)
save_file("kohvimasin.txt", "data/redd/low_freq/house_1/channel_5.dat", times)
```

6 NILM Metadata

```yaml
instance: 1   # this is the first building in the dataset
original_name: house_1   # original name from REDD dataset
elec_meters:
  1: &redd_whole_house
    site_meter: true
    device_model: REDD_whole_house
  2: &emonitor
    submeter_of: 0
    device_model: eMonitor
  3: *emonitor
  4: *emonitor
  5: *emonitor

appliances:
- original_name: lighting
  type: light
  instance: 1
  on_power_threshold: 10
  meters: [2]

- original_name: spaceheater
  type: electric air heater
  instance: 1
  on_power_threshold: 100
  meters: [3]

- original_name: tv
  type: television
  subtypes:
    - smart
  instance: 1

  meters: [4]

- original_name: coffee
  type: sockets
  instance: 1
  meters: [5]
```

7 Evaluation Script

The script used for evaluating the accuracy of different NILM algorithms can be found in the included ZIP archive as evaluation.py.

8 Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, Martin Jürgel,

(autori nimi)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose Non-Intrusive Load Monitoring in the OpenHAB smart home framework,
   (lõputöö pealkiri)

mille juhendaja on Jakob Mass,

(juhendaja nimi)

reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.

4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Martin Jürgel

10.05.2019