MODEL-DRIVEN ANALYTICS OF ENERGY METER DATA IN SMART HOMES

A Dissertation Presented

by

SEAN BARKER

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2014

Computer Science

MODEL-DRIVEN ANALYTICS OF ENERGY METER DATA IN SMART HOMES

A Dissertation Presented

by

SEAN BARKER

Approved as to style and content by:

ABSTRACT

MODEL-DRIVEN ANALYTICS OF ENERGY METER DATA IN SMART HOMES

SEPTEMBER 2014

SEAN BARKER

B.A., WILLIAMS COLLEGE

M.Sc., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Prashant Shenoy and Professor David Irwin

The proliferation of smart meter deployments has led to significant interest in analyzing home energy use as part of the emerging 'smart grid'. As buildings account for nearly 40% of society's energy use, data from smart meters provides significant opportunities for both utilities and consumers to optimize energy use, minimize waste, and provide insight into how modern homes and devices use energy. Meter data is often difficult to analyze, however, owing to the aggregation of many disparate and complex loads as well as relatively coarse measurement granularities. At utility scales, analysis is further complicated by the vast quantity of data, which precludes the use of computationally intensive techniques when monitoring hundreds or even thousands of homes.

In this thesis, I present an architecture for enabling smart homes using smart energy meters, encompassing efficient data collection and analysis to understand the behavior of home devices. I consider four primary challenges within this domain: (1) providing low-overhead data collection and processing for many devices, (2) designing models characterizing the energy use of modern devices, (3) using these models to track the real-time behavior of known devices, and (4) automatic identification of unknown devices in the home.

To enable practical smart homes, my proposed architecture combines low-cost, off-the-shelf sensing equipment with a hybrid local and cloud-based processing backend. To analyze data within the environment, I first characterize the basic device types present in today's homes (e.g., resistive, inductive, or non-linear) and distill the essential usage characteristics of each type. Using these characteristics, I construct a set of models that more accurately represents real-world devices than previous simplistic models. I then leverage this modeling framework to track the behavior of specific devices, using a technique that runs in close to real-time and can scale to many devices. Finally, I present a technique to automatically identify unknown devices attached to smart outlets in homes, which relieves homeowners of the need to manually describe devices in order to employ smart home optimizations.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

The management of energy in today's environment of ever-increasing demand is an issue with profound economic, environmental, and social impact. As homes and office buildings represent over 70% of total grid electricity consumption today [14], significant attention has been paid to 'smart grid' initiatives focused on reducing or optimizing the energy consumption of buildings. This thesis discusses challenges involved in designing energy-aware smart homes and proposes a model-based approach to tackling these challenges.

1.1 Background and Motivation

Computing for sustainability—where real-world physical infrastructure leverages sensing, networking, and computation to mitigate the negative environmental and economic effects of energy use—has emerged as an important new research area. As a result, in addition to improving the energy efficiency of information technology (IT) infrastructure such as mobile devices, servers, and data centers, computing researchers are expanding their focus to now include building energy efficiency. Since buildings account for nearly 40% of society's energy use [32], compared to an estimated 1-2% for IT infrastructure [39], the emerging 'smart grid' has the potential to make a significant impact on society's energy footprint. Managing electricity is particularly critical because buildings consume the vast majority (73%) of their energy in the form of electricity [32].

Existing management techniques typically employ *sense-analyze-respond* control loops: various sensors monitor the building's environment (including electricity) via a smart meter, and transmit collected data in real-time to servers, which analyze it to reveal detailed information about building devices and occupants, and finally respond by automatically controlling electrical loads to optimize energy consumption. However, research challenges exist at each stage of the control loop. Sensing at neighborhood or utility-scales is complicated by the need for prohibitively large numbers of sensors and the associated cost, invasiveness, and upkeep of such deployments. Analysis is often difficult due to the desire to extract fine-grained information (e.g., the behaviors of individual devices) from the relatively coarse-grained data that is generally available (e.g., as from a typical smart meter). When employed by utilities, analysis techniques are further complicated by the need to process large quantities of meter data efficiently. Finally, responding to observations made from analyses is often complicated by practical concerns, such as the presence (or lack thereof) of actuating capabilities within homes themselves. All of these challenges must be addressed to truly design sustainable smart buildings capable of influencing and managing their energy footprint.

This thesis addresses many of these challenges by combining a flexible architecture for efficient energy data collection and processing in buildings with a set of analytic techniques for interpreting this data. In particular, we focus specifically on the case of residential homes, as homes tend to exhibit similar devices, patterns, and opportunities for energy optimization. In particular, we consider the following set of key challenges concerning the design of smart homes:

1. How can we *enable* energy-aware smart homes capable of data collection and analysis without excessive instrumentation overhead?

2. How can we *understand* the energy usage of a home as a function of the many individual devices that make up a home?

3. How can we *apply* this understanding to monitor and analyze the dynamic behavior of smart homes?

1.2 Contributions

In this thesis, I present an architecture for energy-aware smart homes and a set of model-driven techniques for analyzing smart meter data in such homes. The primary components and contributions of this thesis are as follows:

- *Architecture*: A general-purpose architecture for monitoring and processing the energy use of a home environment, focusing on low-cost, off-the-shelf home automation (HA) devices to enable pervasive energy sensing.

- *Modeling*: A modeling framework enabling efficient, device-accurate models of individual loads, based on an understanding and analysis of the essential load types and characteristics of nearly all devices in a typical home environment.

- *Tracking*: A technique that employs the aforementioned modeling framework to accurately track the behavior of individual, previously identified loads within smart homes, in effect providing a 'virtual energy meter'.

- *Identification*: A technique that automatically identifies devices attached to 'smart outlets', thereby relieving homeowners of the need to maintain a mapping of devices to attached outlets in the home.

Each component is described in more detail below.

1.2.1 Architecture

In order to achieve energy optimizations in practice, homes must be able to both collect and manage data about their own environments. The primary challenge here is providing these capabilities without excessive overhead, both in terms of equipment and installation costs (i.e., beyond what a typical 'dumb' home would contain). I describe the design of a smart home architecture centered around an off-the-shelf home energy meter, potentially coupled with smart outlets for finer-grained data collection. In particular, I explore the use of low-bandwidth home automation (HA) devices for fine-grained energy sensing. Data processing is split between a low-power home controller and remote servers for long-term storage and more intensive processing. Results of data analysis are returned to the home in order to perform optimizations (e.g., reducing peak electricity use). Several real-world smart homes following the proposed design comprise an ideal environment for the development and evaluation of data analytics, as described below.

1.2.2 Modeling

The increasing availability of smart meter data in homes has led to significant interest in analyzing this data to understand device behavior, occupant behavior, and key contributors to overall energy use. As raw energy data is often difficult to interpret without prior knowledge of how devices actually operate, most analysis techniques rely on models of device behavior to interpret data from smart meters. Accurate modeling is particularly important when the only data available is aggregated, as is typically the case with a single energy meter providing energy data from the entire house.

Most existing models, however, have been based on overly simplistic assumptions of device operation. For example, most prior work assumes a device will turn on, consume a stable level of power (or possibly one of several stable levels), and then turn off. Through a detailed study of real-world device energy use, I demonstrate that many types of devices do not consume energy in this straightforward way, and therefore that more complex models are necessary to present truly *device-accurate* models. I present a compact but expressive set of models based on fundamental electrical characteristics (e.g., resistive, inductive, or non-linear loads) and demonstrate how nearly all significant devices in today's homes can be represented by one or more of the fundamental model types. Furthermore, I demonstrate how these models increase accuracy and enable more sophisticated analyses of real-world smart meter data when compared to simpler models.

1.2.3 Tracking

Many energy optimizations that consumers and utilities would like to employ (e.g., automatically shifting the operation of specific devices to decrease peak electricity consumption) require relatively up-to-date data from individual devices. For example, an alert system that notifies homeowners of specific events of interest (e.g., a dryer completing its cycle) requires *real-time* data from the device in question, either through a dedicated energy meter attached to the device, or a way to extract this data from aggregated smart meter data. However, existing analysis techniques for extracting individual devices from aggregate meter data (typically known as non-intrusive load monitoring) are not designed to operate online, and are therefore unsuitable for these types of applications.

To address this issue, I present PowerPlay, an efficient, *online* approach to tracking the behavior of specific devices within a home, based on detecting recognizable features from the device models previously proposed. The design of the features and detectors employed allows for online operation on a low-power machine such as in an individual home, or on a server when handling a utility-level quantity of data (e.g., hundreds of homes simultaneously). In doing so, PowerPlay effectively provides the illusion of a 'virtual' smart meter without actually requiring the installation of additional meters. Furthermore, PowerPlay is highly resilient to background devices operating within the same home, which preserves its effectiveness even when many other (potentially unknown) devices are installed in the home. Using our smart home deployments, I demonstrate the accuracy, scalability, and efficiency of PowerPlay on real smart meter data.

1.2.4 Identification

In homes containing smart outlets capable of individually monitoring connected devices, the tracking problem described above is less significant – as individual device data is available directly, there is no need to extract this data from the whole-house data stream. However, this environment introduces new challenges – in particular, in order to effectively make use of smart outlet data, the nature of the attached devices should be known (e.g., type of device, device model, etc). Traditionally, this issue is dealt with by manually maintaining a mapping of outlets to devices (e.g., outlet #6 is attached to the kitchen refrigerator). Maintaining this mapping, however, is difficult for the hundreds of devices that may be found in homes, and error prone when attached devices change (such as a wall outlet that may have a vacuum cleaner or laptop charger attached at different times).

To negate the need to maintain an outlet-to-device mapping, I present a technique for performing 'non-intrusive load identification' (NILI) – that is, automatically identifying the device attached to an outlet based on the energy data from that outlet. By training a classifier to detect specific device types and periodically processing the most recent data from a smart outlet, a dynamic outlet-device mapping may be automatically maintained without manual intervention.

Note that the NILI problem is essentially the inverse of the tracking problem described previously; for tracking, the device of interest is known but its energy use is not, while for identification, the energy use is known but the device itself is not. Both problems are of interest depending on the level of smart home instrumentation (i.e., whether smart outlets are present in addition to the whole-house smart meter).

1.3 Outline

Chapter 2 provides a summary of related work in the areas of smart homes and energy metering. In Chapter 3, I present the flexible smart home architecture enabling energy-based optimizations and summarize the implementation of a real deployment following this design. In the context of this data-driven architecture, Chapter 4 then presents a modeling framework for describing the energy use of commonplace home devices. Using this framework, Chapter 5 describes a technique for tracking specific devices in a smart home using only data from a whole-house energy meter. Chapter 6 addresses the challenge of automatically identifying devices attached to individual smart outlets when such outlets are available. Finally, Chapter 7 provides an overview of this work and summarizes future directions.

CHAPTER 2
RELATED WORK

This chapter presents a survey of background work in smart homes, modeling, and non-intrusive load monitoring (NILM).

2.1 Smart Home Monitoring and Control

Smart buildings use *demand-side energy management* to self-regulate their energy footprint to reduce overall energy consumption and peak power usage, while better aligning consumption with renewable generation [54]. Demand-side management requires buildings to 1) *continuously monitor* the power consumption of electrical loads and 2) *remotely control* when and how much power each load consumes, e.g., to enable automated demand response. Although load monitoring and control are closely coupled in demand-side management, two disjoint sets of technologies have evolved to perform these tasks in smart homes.

On the monitoring front, researchers have developed numerous techniques to enable fine-grain tracking of electric usage at various spatial and temporal dimensions, e.g., [19, 23, 29, 35, 42]. Several past efforts focus on outlet-level monitoring, and in some cases control, of electric loads using wireless technologies, such as 802.15. These technologies are still expensive for typical homes, which may contain several tens to hundreds of outlets. For instance, while not commercially available, ACme meter components cost $85 plus time for assembly and calibration [28, 40], while Tweet-a-watt components cost $60 per meter plus time for assembly [56]. Similarly, the commercial PloggSE plug meter costs $215 per outlet [48]. Other problems include upkeep, communication interference, and aesthetic concerns when deploying large numbers of meters [24]. A less expensive option is to deploy a single sensor at a home's electric panel to monitor aggregate usage, and use NILM or load tracking techniques to disaggregate individual loads. However, this approach alone is not sufficient for *developing* such techniques, as it lacks the capacity for ground-truth data necessary for evaluation. The collection of high-quality datasets has attracted the interest of many researchers in sustainability [37, 2, 7] and represents another goal of smart home deployments.

On the control front, Home Automation (HA) protocols, such as X10 and Insteon [26, 27], were designed explicitly for remote load control. The protocols enable programatic actuation of outlets and switches hard-wired into a building and controlled from a central server, via command-line or remote web/smartphone interfaces, using the building's powerlines for communication. HA protocols are also mature standards: X10 was introduced in 1975 and Insteon in 2005. Many of these protocols communicate over building power-lines, which has been proposed as a path towards smart buildings [44]. Increasingly, monitoring and control capabilities can be combined via 'smart outlets' such as the Belkin WeMo Insight Switch [59], which integrate sensing and actuation capabilities into a single outlet-like device. We consider the challenge in Chapter 3 of augmenting HA protocols for high-resolution monitoring, which requires compensating for very low bandwidths provided by these protocols. The use of HA devices for outlet- and switch-level energy monitoring supplements our proposed smart home architecture, which is based around a whole-house smart meter and low-power controller for performing local data processing. Notably, in contrast to HA protocols, many past efforts focus on outlet- and switch-level energy monitoring using wireless technologies, such as Z-Wave, ZigBee, and WiFi. WiFi is especially attractive due to existing WiFi networks and relatively high bandwidth for fine-grained monitoring. Unfortunately, since outlet and switch boxes are embedded in walls and may be behind large appliances, wireless communication often exhibits interference that severely degrades its performance and reliability.

2.2 Device Modeling

While recent work targets modeling for specific appliances, e.g., a particular brand of refrigerator [54] or an HVAC chiller [9], it does not generalize to a broad range of devices. While the problem of modeling devices is not exclusively useful in NILM or load tracking applications, most prior work in modeling devices has been in the context of performing NILM.

All NILM approaches inherently use some form of load models for devices. However, most existing approaches employ generic on-off load models that represent devices as simple 'step' functions. These techniques typically use these simple models to either i) detect changes in load power states by observing changes in building power, [60] or ii) use Viterbi-style algorithms [17] to determine the most likely set of "hidden" states, e.g., combinations of power states for multiple loads, from a sequence of changes in building power [34]. The simplest edge-detector approaches [22, 23] represent devices using only two states – on and off.

The modeling work presented in Chapter 4 targets a more expressive modeling framework based on understanding the electrical properties of common devices. As nearly all devices fall into a few categories describing their electrical characteristics (e.g., heating devices are resistive, motors are inductive, etc), I detail a compact yet comprehensive set of models that is more accurate than prior models based on small numbers of static states. This work represents a novel application of domain-specific knowledge (i.e., fundamental properties of power systems) to the problem of modeling appliances in homes.

2.3 Non-Intrusive Load Monitoring

One of the highest profile types of smart meter data analysis is termed *non-intrusive load monitoring* (or NILM), which refers to decomposing a house-level smart meter trace into its component devices. Performing this decomposition provides useful information such as (i) the percentage of energy use attributable to certain devices or class of devices, and (ii) the operational patterns of specific devices. There has been over two decades of work on NILM and recent surveys describe and contrast these various approaches [4, 18, 60, 61]. The earliest approaches to NILM involved simple edge-detection (i.e., matching *on* with *off* steps within a meter trace) [22, 23], while more recent techniques use more sophisticated machine learning techniques such as Factorial Hidden Markov Models [34, 37].

A key determinant of the type of technique that is best suited to perform NILM depends on the sampling resolution of the energy meter. Prior approaches have targeted a range of sampling resolutions by using meters that produce 10 million readings every second [45] to 1 reading every hour [38]. Clearly, sampling the total power usage thousands or millions of time every second produces very clearly discernible "signatures" when each device turns on, allowing detection and identification of individual loads [45]. As the sampling resolution becomes coarser, the readings represent the mean power usage over larger and larger intervals, making it harder to discern the turning on and off of individual loads. For instance, at a one hour granularity, it is nearly impossible to detect that a light bulb was switched on for a few minutes during that hour—since the reading indicates the average power usage of a larger collection of loads that were active over the entire hour, making it harder to discern individual loads. Today's utility-grade smart meters provide minute-level sampling (a reading every 5 minutes is typical) but there are indications that the next generation of smart meters will provide sampling resolution of seconds [57].

The work in load tracking presented in Chapter 5 differs from NILM in a few keys ways, most notably by focusing on *online* operation and accurate per-load disaggregation (i.e., effectively providing the abstraction of a 'virtual' smart meter) . Most existing NILM techniques are designed to operate offline over large ranges of data (e.g., 1 day) and output a *complete* energy breakdown, which is difficult due to the number of distinct devices (often 100+) in the home. Furthermore, the per-device accuracy of NILM often suffers when considering fine-grained timescales. The approach to tracking loads detailed in this work explicitly does *not* attempt to account for every load in the home, but instead focuses on efficient, accurate extraction of a small number of devices of interest in an online fashion.

2.4 Device Identification

Prior work on device identification has largely been in the context of whole-house metering, and thus falls under the umbrella of NILM. Device identification from outlet-level meters, or 'non-intrusive load identification' (NILI), however, has received substantially less attention. Accurate load identification has been achieved using high-frequency smart meters [51], but this granularity of data is not typically available from off-the-shelf smart outlets. For lower frequency data, e.g., 1 Hz, the use of classifiers for learning device labels has been proposed, but not extensively evaluated [63, 52], particularly for previously unseen devices. Other approaches to device classification have focused on explicit per-device training to generate 'signatures' that can be used to detect devices in the future [53]. These techniques, however, rely on user-guided training periods to recognize specific devices, as opposed to transparent recognition of devices classes. In this work, we investigate the identification of devices using typical meter data (i.e., 1 Hz) and consider both specific device identification (i.e., a specific model of refrigerator) as well as general device type identification (i.e., any type of refrigerator).

CHAPTER 3
SMART HOME ARCHITECTURE

Any smart home requires two fundamental components: *monitoring*, in which data about electrical usage and any other relevant environmental data is collected, and *analysis*, in which this data is used to derive insights into the home's energy use. An optional third component is *control*, which refers to the ability of the home to automatically actuate loads. While control enables the implementation of automatic energy optimizations, such as rescheduling appliances, it is not necessary for other types of applications, such as user recommendations and many types of analytics.

In this chapter, we describe a smart home architecture encompassing monitoring and analysis, which provides a representative environment in which to explore smart home applications. In particular, we discuss the challenges of providing high-resolution energy monitoring using widely available home automation (HA) products, such as Insteon devices, which allows for adding monitoring capabilities onto existing control capabilities in smart homes. We also provide an overview of our own real-world smart home deployment, which was implemented following this architecture and has been live for over three years. Data gathered from our deployment provides a virtual laboratory for the analytics discussed in Chapters 4 through 6 of this thesis. While actuation and end-user applications (e.g., device scheduling policies) are also important components of real-world smart homes, this thesis focuses only on data analytics, which are necessary precursors to many smart home applications.

3.1 Sensing and Processing Infrastructure

The general-purpose smart home architecture we consider consists of several components within the home, which are pictured in Figure 3.1 and described below.

- A *smart meter* attached to the home's electrical panel. The meter reads the aggregate home energy usage via current transducers and streams power readings every second to the *gateway* (described below). The meter may also collect and transmit other types of data, such as reactive power, frequency, and voltage information. Numerous commercial power meters are available, such as the TED 5000, BrulTech ECM-1240, Current Cost Envi, and eGauge. These meters sense electricity usage using current transducers (CTs) wrapped around each leg of a home's split-phase input power.

- A *gateway* server operating the software platform for the home. This gateway is the 'brain' of the smart home and processes and stores (at least temporarily) data from all sensors in the home. The gateway also performs load analysis on incoming data and load actuation based on customizable optimization policies. While any machine may be used as a gateway server, well-suited choices include 'plug' computers such as the GuruPlug [20], which are compact, low-power, and inexpensive ($100 or less).

- Off-the-shelf *programmable switches* and meters connected to appliances in the home. The gateway communicates with the programmable switches via PowerLine networking, which has been proposed for use in smart buildings [44] and requires no additional infrastructure in the home. Outlet-level meters communicate by PowerLine or wirelessly and provide ground-truth data on individual loads within the home. Both metering and programmability may be provided by all-in-one "smart outlets", which impart these capabilities to any attached de-

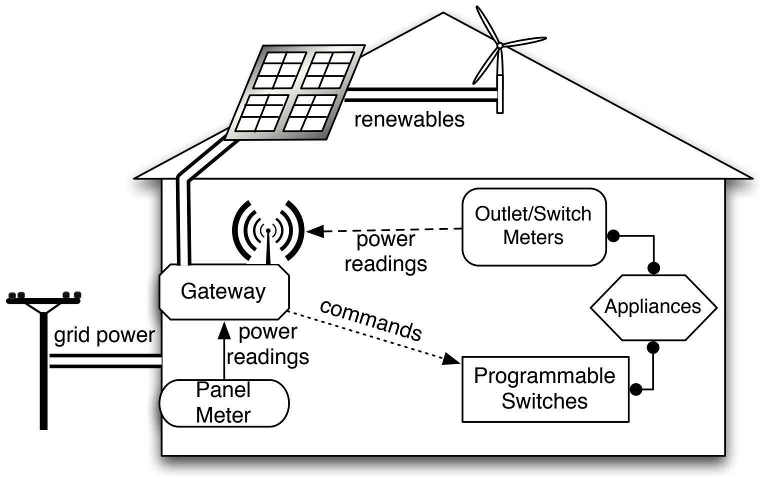


Figure 3.1. In-home architecture of a smart home.

vice. Optionally, other types of sensors may also transmit data to the gateway, such as motion sensors, door sensors, and weather stations.

While the basic components described above are 'complete', the architecture is limited by the capabilities of the gateway server, which will typically be only a low-power machine or even directly integrated into a smart meter. A low-power gateway both limits the processing and data storage capabilities of the smart home. To remedy this issue, we can easily extend the architecture with remote servers accessed over the Internet, as illustrated in Figure 3.2. Here, each home communicates with a central server or set of servers (possibly operated by the utility) and sends data to the central server for long-term storage and analysis. An added benefit of this approach is that the central server may run analytics over multiple homes at once, such as if a utility wishes to calculate the total energy use of its customers' air conditioners. Mechanisms for control within homes themselves, may or may not be accessible, depending on the level of control that homeowners wish to delegate to the utility.

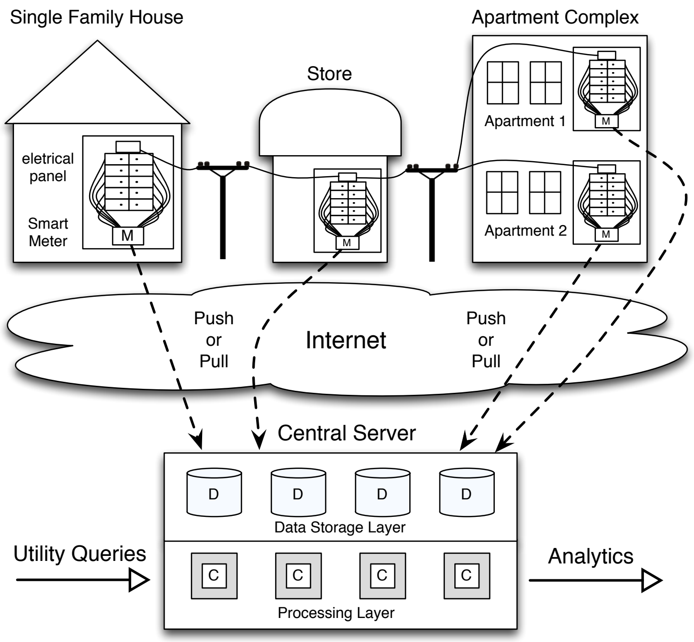


Figure 3.2. A smart home architecture supplemented with a remote, central server for storage and analysis over large data streams.

3.2 Real-World Deployment

We have implemented the smart home architecture detailed in Section 3.1 within three real-world homes in order to drive our research (including, but not limited to, the work detailed in the remainder of this thesis). The most fully instrumented home is described below, while details on the other two are available in [7].

Our primary instrumented home is a two-story, 1700 square foot home with three full-time occupants. The home has a total of eight rooms including its basement. Major appliances include window air conditioning units, an electric dryer, a washing machine, a heat recovery ventilation (HRV) unit, a dishwasher, a refrigerator, and a freezer. The home has 35 wall switches, which primarily control room and closet lighting; switches also control an exhaust fan in each bathroom and the garbage disposal. The electrical panel has 26 individual circuits.

The smart home deployment is centered around a gateway in the form of an embedded Linux server (a GuruPlug) that queries each sensor to collect data. Using an eGauge [15] smart meter installed in the mains panel (pictured in Figure 3.3, we collect electricity data every second for the entire home as well as each individual circuit. We have replaced 30 of the home's 35 wall switches with units that transmit on-off-dim events for the switches to the gateway server. We are able to derive the power usage from the uninstrumented switches via the circuit data: the basements switches are on dedicated circuits, the garbage disposal is on a circuit with only the dishwasher (which has a dramatically different power profile), and the kitchen switch is on a circuit dedicated to kitchen lights, which has only one other already-instrumented load. The home's electrical wiring also aids our data collection. Each circuit is dedicated to either lighting (monitored at wall switches), outlets (monitored by plug meters – in effect providing smart outlets), or individual large appliances (monitored at the mains panel). Our plug-level sensors are commodity Insteon iMeters [25] and Z-Wave Smart Energy Switch meters [1], which use powerline and wireless communication, respectively, to transmit readings to our smart home gateway. Since our wall switches report on-off-dim events, rather than raw power, having the lighting on separate circuits makes it simple to correlate lighting events with power usage using the circuit data. In addition to local storage on the gateway server, data from eGauges and wall outlets is also transmitted nightly to a central server (as detailed in Section 3.1), where it is archived for later retrieval and analysis.

As our analytic work is highly dependent on the quality of the energy data collected by our deployment, we have conducted experiments to verify the accuracy of our meters. The primary eGauge meter is rated to have less than 1% error for current and voltage. Redundant monitoring of both the home's aggregate data and every circuit allows us to determine the relative error of the sensors, by comparing the aggregate usage with the sum of all circuits' usage. Figure 3.4 demonstrates that over 90% of the per-second readings for the entire home and the sum of the circuits is within 2% of each other, while over 99% of readings are within 4% of each other.

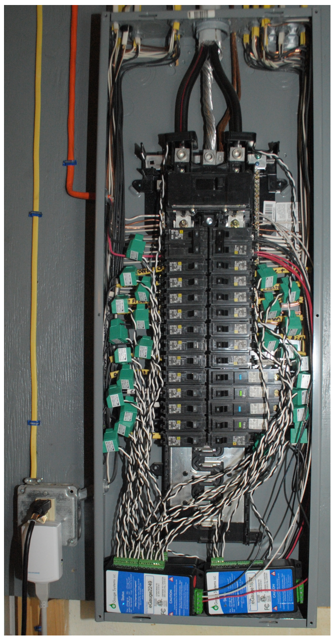


Figure 3.3. CT installation at the mains panel.

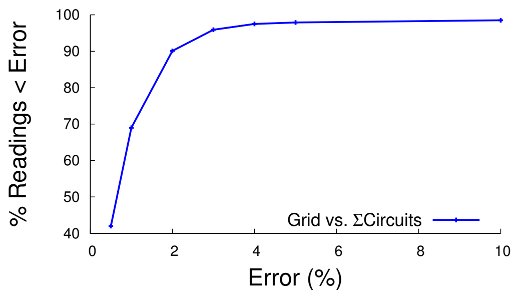


Figure 3.4. Error between Home A's aggregate electricity data and the sum of all the individual circuits

In addition to monitoring power usage in the home, our infrastructure has support for tracking other metrics, such as solar generation and environmental data from weather sensors (temperature, wind speed, etc.). Additional sensors also track events within the home, such as motion detectors, door sensors, and thermostat sensors.

Our system has been continually monitoring hundreds of individual loads every second for over three years in each of our three instrumented homes. Since our level of instrumentation is time-consuming and expensive to replicate, we have made much of our collected data available to benefit other researchers [7]. To date, this dataset has been downloaded by over 600 unique users in over 80 countries. This data has also enabled our own smart home research, such as the analytic challenges discussed in Chapters 4 through 6.

3.3 Combining Monitoring and Control

Many of the "smart home" devices available today (e.g., remotely controlled switches and sensors), as well as much of our own deployment, fall under the umbrella of "home automation" (HA). Once designed for hobbyists, HA has now become mainstream and is widely used in smart buildings for actuation of outlets and switches, using a building's powerline for communication via protocols such as Insteon. These protocols have a number of advantages that make them attractive for use in smart homes, including widespread commercial availability, open standards, compatibility with existing buildings using HA protocols, and the high reliability of the powerline communication typically used. Despite these benefits, however, using HA-based protocols for energy monitoring poses significant challenges:

Scalability. HA protocols were not designed to support continuous monitoring traffic and often lack basic features such as collision avoidance, making high-resolution energy monitoring challenging for even a small set of devices. Thus, a key research challenge is *scaling* HA protocols to monitor large numbers of devices.

Accuracy. HA protocols are capable of monitoring power state changes for switches and low resolution power usage for outlets. Thus, another key challenge is *accurately* translating switch state change events and coarse outlet power data into high resolution power usage measurements.

As many of our own outlet-level meters are Insteon devices, we have tackled this challenge by designing "smart polling" techniques to enable reliable, high resolution power monitoring using HA protocols. Here we provide an overview of the Insteon protocol as well as our polling techniques to enable high-resolution sensing in smart homes using Insteon hardware.

3.3.1 Insteon Protocol

To support monitoring, Insteon-enabled wall switches send asynchronous *notifications* whenever someone toggles the switch, while the Insteon-enabled outlets must be explicited *queried* for their power usage. The primary difficultly in using HA protocols such as Insteon for energy monitoring is their extreme bandwidth limitations. While more recent powerline-based protocols, such as HomePlug, provide much more bandwidth, they are not used for HA, and instead are targeted at high-bandwidth data from Internet traffic. HomePlug is not typically embedded into standard wall outlets and switches due to both cost and form factor constraints. We focus on Insteon in our deployment, since it is an extension of the original X10 HA protocol with greater reliability and higher bandwidth.

In the Insteon protocol, senders broadcast messages over a building's powerline, while receivers listen for messages and send acknowledgements upon receipt. All Insteon devices also act as repeaters that automatically repeat messages they hear a fixed number of times, alleviating the need for complex routing protocols to transfer messages. The protocol also avoids flooding and collisions when repeating messages, since all devices synchronize retransmissions using the 60Hz AC powerline frequency—each transmission begins exactly 800 microseconds before the zero crossing and ends exactly 1023 microseconds after the zero crossing.

The Insteon protocol supports two types of messages: 10 byte standard messages and 24 byte extended messages, which require 6 and 13 zero crossings to transmit, respectively. Since there are 120 zero crossings per second with 60Hz AC power, a standard message takes 50ms to transmit and an extended message takes 108.33ms, with no additional hops. While Insteon's maximum theoretical bandwidth is 2880bps, in practice, messages typically travel three hops and return acknowledgements, which reduces the maximum bandwidth by 16X to 180bps. In addition to repeated messages, a sender that does not receive an acknowledgement within a timeout will retransmit a message up to five times. Thus, actual bandwidth may be much less than 180bps with three hops.

Finally, note that Insteon *does not* prevent multiple devices from sending different messages at the same time. While repeating messages avoids flooding and collisions due to the synchronized retransmissions, Insteon has no collision avoidance mechanism (likely due to its design towards low bandwidth control). At high message rates, Insteon's lack of backoff combined with its static number of multiple hops and retransmissions per message results in repeated collisions, causing bandwidth to abruptly collapse. Setting the query rate for outlets presents a tradeoff: a rate too high will saturate the available bandwidth and result in the loss of either asynchronous switch notifications or control commands, while a rate too low will result in coarser and less accurate outlet power data. To understand this tradeoff, we experiment with our smart home deployment by varying the rate of outlet queries, and then determining both the percentage of queries lost (Figure 3.5) and the percentage of switch notifications lost. For each data point, we issue outlet queries at the specified interarrival time on the x-axis for 10 minutes, while turning wall switches on and off 50 times, such that the time between toggling the switch is uniformly random between 0 and 20 seconds. We also perform a similar experiment in isolation in a separate building with no other devices attached to the powerline.

Each outlet query includes three standard Insteon messages and one extended message: a standard query message from PLM $\rightarrow$ outlet, an extended response message from outlet $\rightarrow$ PLM with the outlet's current average power usage, and a standard acknowledgement for each message. Based on the protocol specification, each outlet query should take 4*(0.05+0.05+0.1083+0.05) = 1.0333 seconds, including the original message and the three additional hops.

Below, we use the specification to model the percentage of outlet queries we expect to receive, and the percentage of switch notifications we expect to lose for different query rates. We construct a simple model of the probability of losing a switch notification (S_{lose}) as a function of the interarrival time of outlet queries (T_i) and the length of an individual query ($T_q = 1.0333$). For simplicity, our model assumes that when transmissions collide, the transmission from the device physically closer to the PLM is successful. In addition, our model assumes one retransmission due to powerline noise, and no additional retransmissions due to collisions. Thus, the model divides the length of an individual query (T_q) by two times the interarrival time between queries (T_i), where the factor of two in the denominator approximates the effect of one extra retransmission.

$$S_{lose} = \frac{T_q}{2 * T_i} = \frac{0.5166}{T_i} \tag{3.1}$$

We also model the probability of receiving a query $Q_{receive}$ as the function below. If we issue queries at an interval greater than the query length, then we expect to receive every query. For intervals less than the query length, we expect queries to increasingly collide.

$$Q_{receive} = \begin{cases} 1 & : T_i > 1.0333 \\ \frac{T_i}{T_q} = \frac{T_i}{1.0333} & : T_i < 1.0333 \end{cases}$$

Figure 3.5 shows that, as expected, issuing queries faster then the 1.0333 seconds it takes to complete them rapidly degrades performance. In isolation, our results show an abrupt drop in the percentage of outlet queries received once the interarrival time hits the protocol's saturation point at 1.0333 seconds. In isolation, the actual drop is more sudden than our model, since the model does not account for multiple

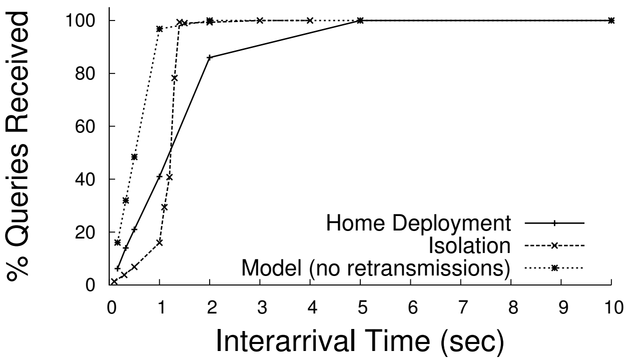


Figure 3.5. Insteon does not support high outlet query rates.

retransmissions of a lost message, which immediately collapses the available bandwidth. Our smart home deployment also shows more query losses than our model before the saturation point, which is likely due to i) additional losses from powerline noise due to other devices and ii) collisions with switch notifications and the resulting retransmissions.

Our results highlight the limitations of using Insteon for monitoring the energy usage of many devices. Consider a simple approach to querying outlets that issues one query every 10 seconds to a new outlet in round-robin fashion. Thus, given N outlets in a building, this approach can query each outlet once every $10 * N$ seconds. Since our home deployment currently uses thirty Insteon-enabled outlets, we are able to measure each outlet's power once every five minutes.

A five minute data resolution is not effective at monitoring the energy usage of most types of devices. As we show in prior work [6], many non-linear electronic devices, such as LCD televisions, exhibit rapid and significant changes in power, e.g., >100W every second, when turned on. Other high-power resistive and inductive devices also exhibit complex patterns of power usage that change every second. Further, since many devices, such as a microwave or toaster, have operating times much less five minutes, this approach cannot detect their operation. In fact, the only devices this simple approach can accurately detect are low-power resistive devices, which exhibit highly stable power usage, that are left on for more than five minutes. The only prominent low-power resistive devices are incandescent light bulbs, which are slowly being phased out. Of course, additional outlets will further decrease the power data resolution.

In addition, even when employing such a low query rate, the probability of losing a switch notification or a control command is still near 5%. Since the OS issues control commands, losing them does not present a significant issue, since it can recognize their loss at application-level and resend them. However, our simple approach to monitoring would have no way to detect a lost switch notification, so it will miss 5% of them at this query rate. Thus, our results motivate a more efficient approach to to monitoring outlet power usage that judiciously controls the number of outlet queries.

3.3.2 Smart Outlet Polling

Insteon-enabled outlets must be **polled** in order to read the energy use of the attached device, which results in many devices competing for limited global bandwidth. The simplest polling approach is to continuously query Insteon-enabled outlets in a round-robin fashion at a static query rate. While simple, this approach suffers the most from bandwidth limitations, since devices that rarely change state (such as lights) are polled at the same rate as highly variable devices (such as a washing machine). Instead, we employ several "smart polling" techniques to make better use of bandwidth, as described below.

Frequency-based polling. The first technique we consider is a modification of round-robin polling in which devices are polled at different rates. Highly active devices may be polled more frequently in order to more accurately capture their behavior, while mostly static devices may be polled infrequently. Here, we define a device's level of 'importance' as its frequency of energy state changes (i.e., power increases or decreases) that the device exhibits over a typical day. Given the state change frequency for each outlet, our system polls each outlet at a rate proportional to its frequency, scaled such that the system continuously polls at a fixed global query rate (as in round-robin).

Event-driven polling. Our second technique makes use of a centralized "smart" meter that monitors power for an entire building at high resolution. Such smart meters are widely available commercially, and are increasingly being installed by utilities: in 2011, nearly 500 utilities in the U.S. had collectively installed more than 37 million smart meters. In our deployment, we use an eGauge meter installed in the electrical panel, which measures building-wide average power usage as well as per-phase average voltage and frequency each second.

Given a centralized meter, event-driven polling analyzes live data from the meter to poll **on-demand** when energy changes (events) occur. Since the meter records aggregate energy, an energy event stemming from any individual device will be reflected in the aggregate data. When an event is detected, a frequency-based polling round of individual outlets is conducted to attribute the event to a specific device, and stopped once the matching energy change is found. As a result, in most cases, only a subset of devices must be polled to identify the source of an event.

3.3.3 Smart Polling Evaluation

To evaluate the three polling approaches, we conduct a simulation study using real-world, device-level data gathered from our deployment. Our sample dataset consists of 24 hours of 1 Hz data from 22 distinct outlets. We replay this data while simulating polling at a variable rate and measure the number of 'events' captured in the resulting monitored traces, where an event is defined as a power change of at least 10W. The percentage of events missed for each of the three polling approaches is shown in Figure 3.6 as the polling interval is varied (here we assume that all polls are completely successful).

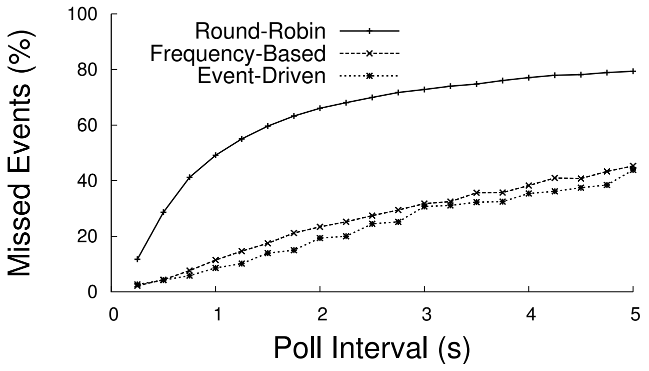


Figure 3.6. Frequency- and event-driven polling capture more energy events than basic round-robin polling.

We immediately see that the naive round-robin approach misses significantly more events (at least 2X) than the frequency-based or event-driven approaches. The poor performance of round-robin is largely due to a small number of highly-variable devices, such as a TV and washing machine. Smart polling compensate for these devices by polling them much more rapidly, which ensures that most events continue to be captured. Notably, frequency-based polling performs nearly as well as event-driven polling despite its lack of a building-wide smart meter.

The key benefit of event-driven polling is that polling may be stopped completely when no events are occurring. Figure 3.7 shows the aggregate bandwidth consumption over the course of the (simulated) day using the event-driven approach. While round-robin and frequency-based polling both poll continuously (effectively using 100% of available bandwidth), the event-driven polling employs far fewer polls — less than 20% — to achieve the high accuracy seen in Figure 3.6. Thus, we expect the event-driven approach to scale well to large number of devices.

3.4 Summary

This chapter presents a general-purpose architecture for designing smart homes that provide robust capabilities for energy monitoring and data analysis. In particu-

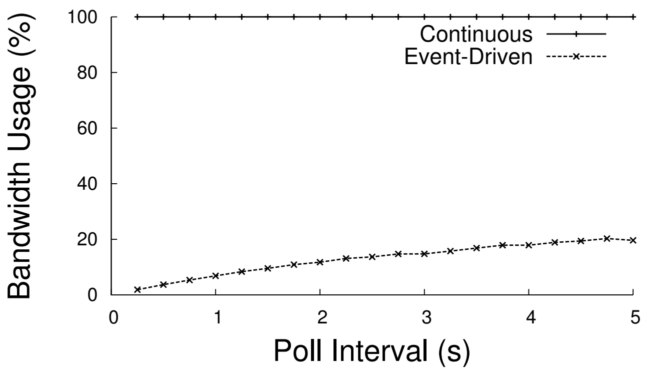


Figure 3.7. Event-driven polling drastically decreases the bandwidth required to monitor a set of outlets.

lar, by combining a whole-house smart meter, a low-power home gateway server, and optionally a set of smart outlets, we can provide an efficient, flexible, and low-cost platform for smart home applications. We also discuss the challenges of providing high-resolution energy monitoring using home automation protocols such as Insteon, which are popular in smart homes but not generally designed for continuous monitoring. We present an approach to leverage HA protocols for monitoring through “smart polling” technqiues, and show that they outperform naive polling approaches in both monitoring accuracy (increased by 2X) and total bandwidth consumption (decreased by 80%).

CHAPTER 4
APPLIANCE LOAD MODELING

The rapid deployment of digital smart meters by electric utilities has resulted in the availability of substantial amounts of fine-grained electricity data for buildings and homes. For example, Pacific Gas and Electric now operates over nine million smart meters in California [46]. While today's deployed smart meters typically measure average power usage at intervals ranging from fifteen minutes to an hour, the granularity of data is trending downwards (e.g., some utilities already provide 5-minute data [47]), and commodity meters are available that measure and transmit, via the Internet, energy usage at intervals as small as every second [15, 55]. Combined with the emergence of "big data" cloud storage systems, these smart meter deployments are spurring renewed interest in analysis techniques for smart meter data. In this chapter, we present a flexible modeling framework for describing the energy use of modern devices that is more accurate than simpler models previously employed.

4.1 Background and Motivation

While prior research has made significant progress in deriving insights from smart meter data [6], one issue that often limits accuracy is the use of general, and often simplistic, load models. In particular, many prior techniques for analyzing and modeling building electricity data characterize loads using simple on-off models, which associate a small number of fixed power levels with the "on" state (often just one) and either no power usage, or some minimal amount, with the "off" state.

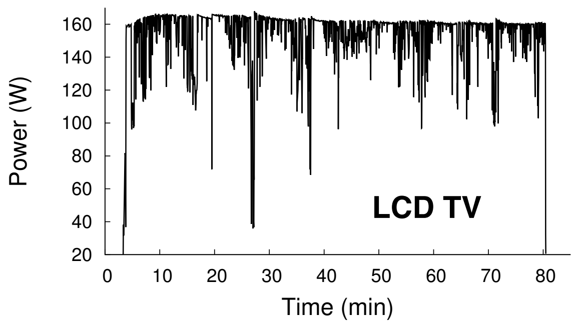


Figure 4.1. An LCD TV's power usage varies rapidly, significantly, and unpredictably while on, and does not conform to a simple on-off load model.

On-off models do have a number of advantages. For instance, they exactly capture many simple loads, including light bulbs and other low-power resistive devices with mechanical switches. In addition, on-off models allow researchers to describe buildings as state machines that associate each building state with a fixed power level (implying the set of loads that are on), and where state transitions occur whenever a load turns on or off. Characterizing buildings as state machines admits a plethora of analysis techniques. For instance, much prior work maps building state machines to Hidden Markov Models (HMMs), and applies HMM-based techniques, such as Viterbi's algorithm [17, 58], to determine which loads are on in each state. In this case, using only a few (often two) power states per load is advantageous, since it minimizes the number of distinct power states for the entire building and reduces the complexity of analyzing the resulting state machine. Of course, even with only two power states per load, the number of building power states is still exponential in the number of loads, i.e., 2^n for n loads. Thus, even assuming simplistic on-off load models, precise analysis may still be intractable, i.e., require enumerating an exponential number of states.

Unfortunately, while on-off load models are simple, they are often inaccurate, since they fail to capture the complex power usage patterns common to many loads. As a simple motivating example, Figure 4.1 shows a time-series of an LCD TV's electricity usage each second. In this case, the TV's switched mode power supply (SMPS) causes power variations as large as 120 watts (W) by rapidly switching between a full-on and full-off state to minimize wasted energy. The magnitude of these variations is effectively random—determined by the color and intensity of the TV's pixels. An on-off model clearly does not accurately capture the TV's power usage. As a result, modeling the TV as an on-off load may complicate higher-level analysis techniques for smart meter data. For example, the TV may obscure the use of low-power loads, such as a 60W light bulb, since its power usage varies rapidly by >60W.

Our premise is that simple on-off models discard a significant amount of information that is potentially useful in analyzing data. As a result, here we focus explicitly on accurately characterizing and modeling a variety of common household loads. Our methodology is empirical: we i) gather fine-grained electricity usage data from dozens of loads across multiple homes, ii) characterize their behavior by distilling a small number of common usage attributes, and then iii) derive accurate load-specific models based on these attributes. One of our contributions is to show that a small number of model types, stemming from basic knowledge of power systems, accurately describe nearly all household loads. Thus, one of our goals is to highlight how many identifiable load attributes, which are well-known in power systems, manifest themselves in electricity data collected by smart meters. *Our hypothesis is that accurate load models, which leverage domain knowledge from power systems, provide a foundation for designing new electricity data analysis techniques.* In evaluating our hypothesis, we make the following contributions.

Empirical Data Collection and Characterization. We instrument a wide variety of common electrical loads in multiple homes, and collect electricity usage data for each load, every second, for over two years. We show empirically that homes operate similar types of loads, e.g., lighting, AC motors, heating elements, electronic devices, etc., which results in significant commonality in power usage profiles across loads. We then characterize the data to identify distinguishing attributes in per-load power usage, forming the building blocks of our models. While many of these attributes are well-known in power systems, we show how they manifest in sensor data.

Data Modeling Methodology. We use our empirical characterization to construct a small number of load-specific model types. We show that our basic models, or a composition of them, capture nearly all household loads. Our models go beyond on-off models, by capturing power usage characteristics that i) decay or grow over time, ii) have frequent variations (as with the TV in Figure 4.1), iii) exhibit complex repetitive patterns of simpler internal loads, and iv) are composites of two or more simpler loads. We show that our models are significantly more accurate than on-off models, decreasing the root mean square error by as much as 8X for representative loads. Since our methodology is general, it is applicable to modeling other types of loads as well beyond those described in this chapter.

4.2 Empirical Data Collection and Characterization

A typical home consists of dozens of electrical loads, including heating and cooling equipment, lights, appliances of various kinds and electronic equipment. A partial list includes:

- *Heating, cooling, and climate control equipment* such as a central air conditioner, window air conditioner, space heater, electric water heater, dehumidifier, fan, air purifier;
- *Kitchen appliances* such as an electric range, microwave, refrigerator, coffee maker, toaster, blender, dishwasher;
- *Laundry appliances* such as a washing machine and dryer;
- *Lighting* including incandescent and fluorescent lights;

- *Miscellaneous electronic devices* such as a television, audio receiver, radio, battery charger, laptop and desktop computer, and gaming console; and
- *Other appliances* such as a vacuum and carpet cleaner.

Since our methodology is empirical, we leverage data collected from our smart home deployments (described previously in Chapter 3) to characterize various loads based on a few elemental types, as described below.

4.2.1 Characterizing Different Types of Loads

Despite their tremendous variety, most residential loads fall into one of a few elemental load types based on how they consume power in an alternating current (AC) system. In particular, loads are categorized as either resistive, inductive, capacitive, or non-linear based on how they draw current in relation to voltage, which in an AC system varies along a smooth sinusoidal pattern. These categories reveal properties of the loads that we leverage in our models. Since many researchers outside of power systems may be unfamiliar with these load types, for each type of load we first review its salient characteristics. We then empirically characterize data from multiple representative loads of each type to observe how their specific characteristics manifest themselves in the data.

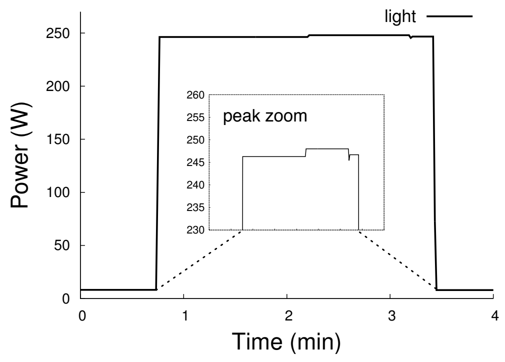
Resistive Loads. Loads that consist of any type of heating element are resistive. Incandescent lights, toasters, ovens, space heaters, coffee makers, etc., are examples of common resistive loads in a home. Formally, if a load draws current along a sinusoidal pattern in the same phase as the voltage, i.e., the maximum, minimum, and zero points of the voltage and current sine waves align, then the load is purely *resistive*.

Figure 4.2 depicts a time-series of the power usage for five different resistive loads with heating elements: an incandescent light bulb, a toaster oven, a coffee maker, a sandwich press, and a pod coffee maker, e.g., a Keurig or Tassimo. In general, the power usage of these loads resembles a “step” when turned on, with usage that remains relatively stable and flat. The incandescent light acts as a nearly perfect resistive load with a power usage equal to the bulb’s wattage. While the toaster oven, coffee maker, sandwich press, and pod coffee maker act similar to the light bulb, they experience an initial higher power usage that slowly decays to a relatively stable usage, highlighted in Figure 4.2. The initial high power is due to the large inrush (or surge) current that occurs as the device warms up and the resistance decreases, after which it stabilizes.

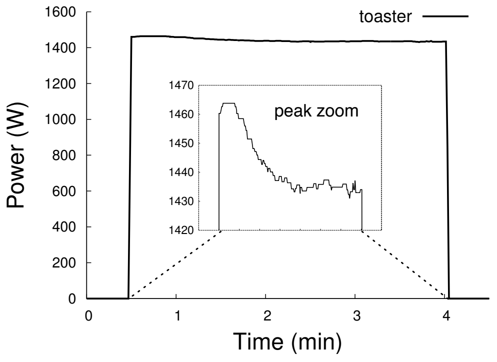
Observation 1: Resistive loads exhibit stable power usage when turned on, with high-power heating elements exhibiting an initial surge followed by a slow decay to stable power.

Inductive Loads. AC motors are the most common and widely-used examples of inductive loads. Motors are the primary component of many household devices, including fans, vacuum cleaners, dishwashers, washing machines, and compressors in refrigerators and air conditioners. Formally, if a load draws current along a sinusoidal pattern that peaks *after* the voltage sine wave, i.e., the current waveform lags the voltage waveform, then the load is purely *inductive*.

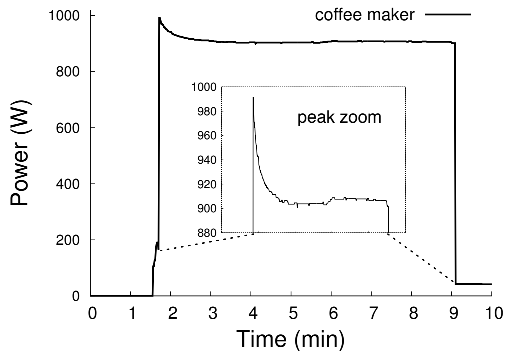
Figure 4.3 depicts a time-series of the power usage for five inductive loads: a refrigerator, a freezer, a central air conditioner (A/C), a vacuum cleaner, and a window A/C unit. All five loads operate AC motors. Unlike the resistive loads above, each inductive load experiences a significant, but brief, initial power usage. The surge is also due to inrush current that occurs when starting an AC motor, although it is typically much higher than for heating elements. Intuitively, the underlying reason is that, while heating elements heat up slowly, the rotor inside a motor must transition from completely idle to full speed within seconds. Power usage then exhibits either a decay or growth, depending on the motor’s operation, that eventually stabilizes. In contrast to resistive loads, motors exhibit small variations even during this “stable” phase. For instance, the refrigerator shown in Figure 4.3(a) exhibits small fluctuations



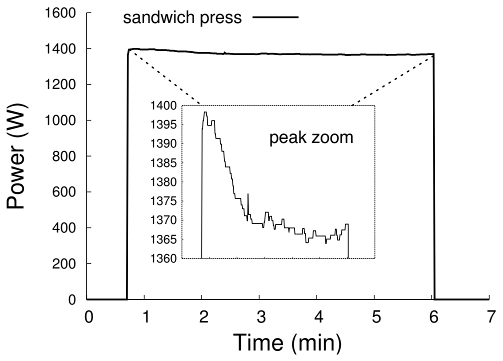
(a) Light bulb



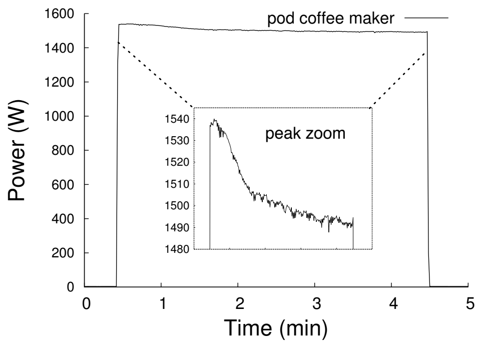
(b) Toaster



(c) Coffee maker



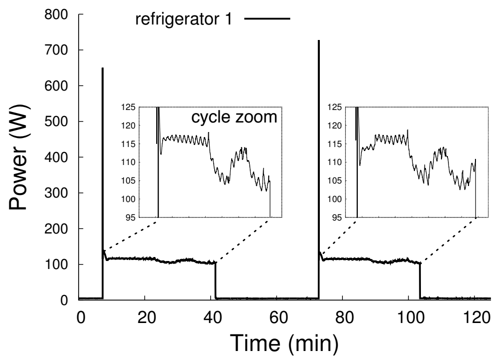
(d) Sandwich press



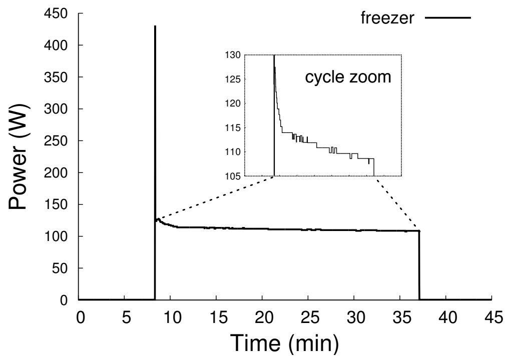
(e) Pod Coffee

Figure 4.2. Example resistive loads, demonstrating "step" behavior with a possible initial surge and slow decay to a stable power level.

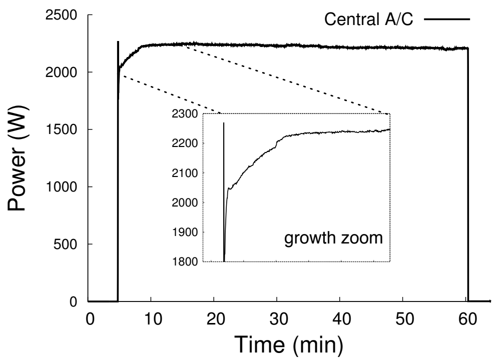
that repeat during each cycle of the compressor. The freezer, central A/C, vacuum cleaner, and window A/C depicted in Figures 4.3(b), (c), (d), and (e) all show an initial spike followed by a sharper, smoother growth (central and window A/C) or decay (freezer and vacuum), with small variations as the usage stabilizes. These patterns



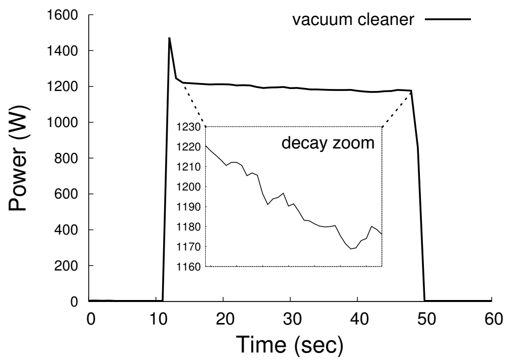
(a) Refrigerator



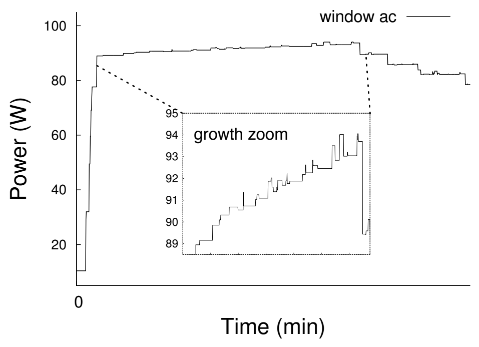
(b) Freezer



(c) Central A/C



(d) Vacuum cleaner



(e) Window A/C

Figure 4.3. Example inductive loads, demonstrating significant surge current followed by steady power growth or decay.

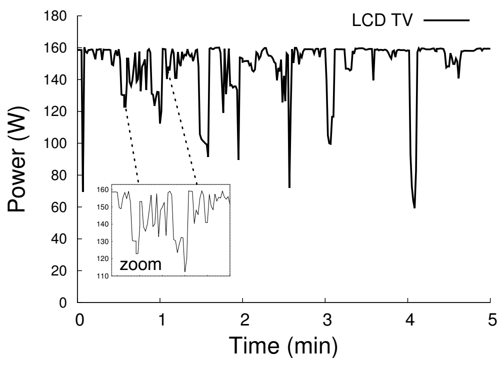
demonstrate that, unlike resistive loads, modeling inductive loads using simple on-off step functions is problematic.

Observation 2: Inductive loads with AC motors exhibit an initial power spike followed by a growth or decay to a stable power level. The growth/decay rate is load-dependent, with the stable power level also exhibiting fine-grained variations.

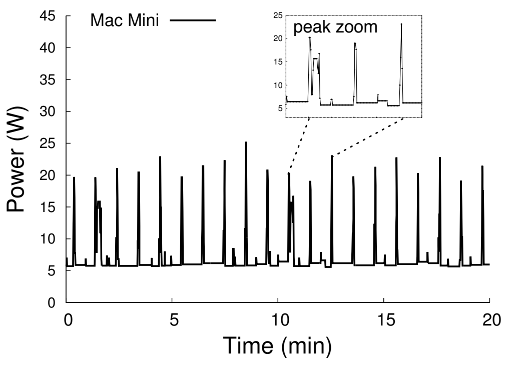
Capacitive Loads. Capacitive loads are the dual of inductive loads. Formally, if a load draws current along a sinusoidal pattern that peaks *before* the voltage sine wave, i.e., the current waveform leads the voltage waveform, then the load is purely *capacitive*. While many loads have capacitive elements, they generally occur in addition to other resistive and inductive elements which dominate their overall behavior. Thus, there are no significant capacitive loads in buildings, particularly when considering real (as opposed to reactive) power.

Non-linear Loads. Finally, any load that does not draw current along a sinusoidal pattern is called *non-linear*. Non-linear loads may also be resistive, inductive, or capacitive based on when their current waveform peaks. The most predominant non-linear (and largely inductive) loads are electronic devices, including computers and TVs. The non-linear nature of these loads is primarily due to the use of switched-mode power supplies (SMPSs). Fluorescent lights are another example of a non-linear (inductive) load. Smaller electronic devices that convert AC to low-voltage DC, such as battery chargers for portable devices and digital clocks, are also non-linear.

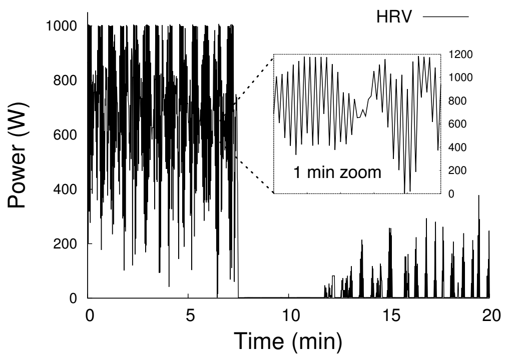
Figure 4.4 shows the power usage of five different non-linear loads: an LCD TV, a Mac Mini desktop computer, a microwave oven, a duct heater for a heat recovery ventilator (HRV), and a computer monitor. These loads exhibit significant power fluctuations when active, but also have a stable floor or ceiling from which these fluctuations derive. The LCD TV shown in Figure 4.4(a) exhibits a stable maximum usage with random power reductions from this ceiling. These fluctuations result from displaying a variety of color and pixel intensities on the screen. Not surprisingly, the computer monitor in Figure 4.4(e) has a similar pattern of power usage. In contrast, the desktop computer shown in Figure 4.4(b) has a stable minimum power draw,



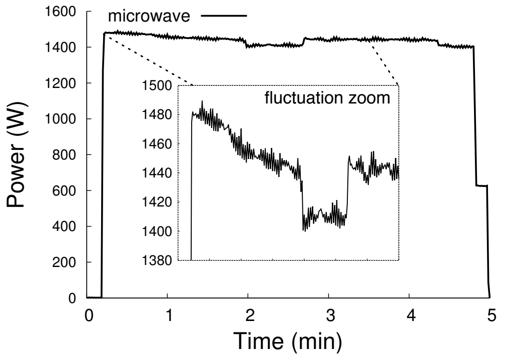
(a) LCD TV



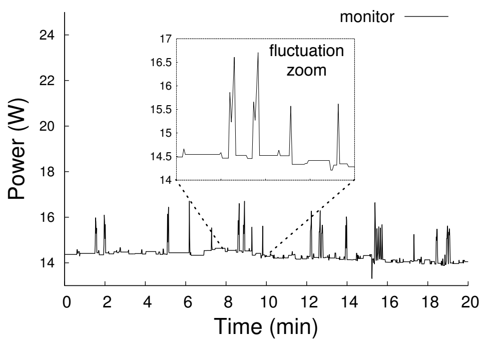
(b) Desktop PC



(c) Duct Heater



(d) Microwave



(e) Monitor

Figure 4.4. Example non-linear loads, demonstrating rapid and significant random variations with possible ceilings and/or floors.

with random power spikes above this floor depending on its workload, e.g., causing the CPU to ramp up, etc. Both the TV and desktop computer consist of a switched mode power supply (SMPS) that regulate the power usage of the device and switch between a full-on and full-off state to minimize wasted energy. The duct heater shown in Figure 4.4(c) demonstrates two regular modes of operation: an active heating mode—with instantaneous intensity managed by the HRV controller—and a passive mode. In both modes, there are large, random variations in power usage. In the active state, there is also a clear stable maximum usage. Finally, the microwave shown in Figure 4.4(d) has what initially appears to be a straightforward step, similar to the resistive loads. However, zooming in shows the microwave's non-linear behavior, with rapid, albeit small, variations in the second-to-second usage, along with larger periodic power shifts. These examples show that on-off models are inappropriate for non-linear loads, since two power states cannot capture their wide range of power variations.

Observation 3: Non-linear loads exhibit significant random variations in power usage. These fluctuations are often range-bound and capped by a floor or ceiling in the power level.

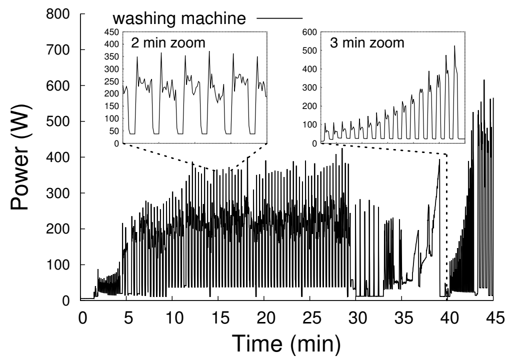
4.2.2 Composite Loads and Reactive Power

Composite Loads. Many household loads, particularly large appliances, are not purely resistive, inductive or non-linear. Instead, these loads consist of multiple components, each of which may be one of the simpler load types. For instance, a central air conditioner may consist of a compressor, a fan to blow air into ducts, duct dampeners to control air flow, and central humidifiers to control humidity. A refrigerator, which has a compressor that is an inductive load, may also consist of door lights, an ice maker, and a water dispenser. Similarly, electric dryers, washing machines, and dishwashers also consist of a motor—to spin clothes and circulate water via a pump—and a heating element—to dry clothes or warm water. In addition, these appliances often operate in repetitive cycles that activate each of their constituent loads differently, such as washing, draining, and then drying for a dishwasher. Figure 4.5 depicts the power usage of a washing machine, a dryer, two dishwashers, and the HRV. As shown, these loads exhibit distinct behavior in different parts of their cycle depending on which appliance component is in use. For example, based on the observations above, distinguishing when a complex load, such as the dryer, activates its heating element versus its motor is straightforward. Finally, an appliance may activate its various components in sequence, in parallel, or both. For instance, a central air conditioner may operate the compressor, the fan and the dampeners concurrently, while a dishwasher may operate its motor, pump and heater in sequence.

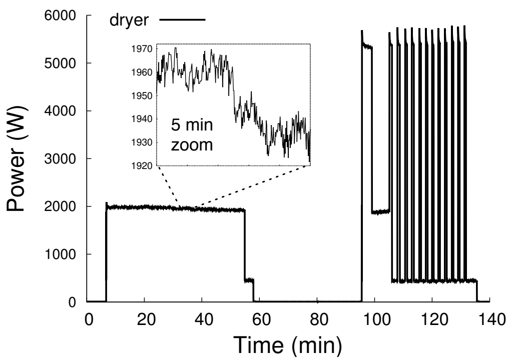
Observation 4: Composite loads consist of simpler resistive, inductive and non-linear loads that operate in parallel, in sequence, or both. As a result, composite loads exhibit distinct behaviors in different operating regions of their active cycle.

Reactive Power. Finally, another important characteristic of the elemental load types above is how they consume *reactive power*. While real power is the amount of power delivered to a load, and is often referred to as simply electricity or power (without the qualifier), reactive power is the amount of power generated, but not delivered, to the load; it is also measured in units of watts, but written as voltage-amperes reactive (VAR) to distinguish it from real power. Reactive power arises when a load draws current *out of phase* with the voltage. Thus, only non-resistive loads generate reactive power. At a high level, reactive power is the result of the instantaneous power (the product of current and voltage) occasionally becoming negative within each AC cycle, due to out-of-phase current and voltage. This state causes power to flow towards the generator and away from the load. Reactive power is typically dissipated as heat in power lines. For our purposes, reactive power provides additional useful information for modeling, and many commodity power meters are capable of measuring it. As a result, our models include both real and reactive power.

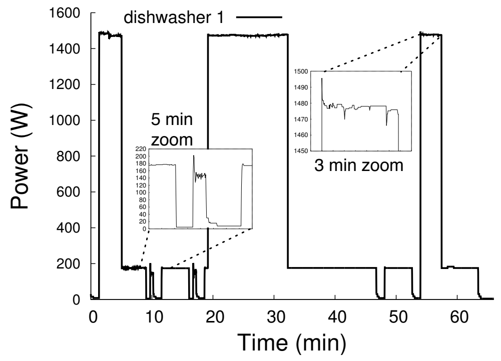
Figure 4.6 depicts companion graphs for selected loads that shows their reactive, rather than real, power usage. Figure 4.6(a) shows that, similar to real power, a resistive dimmable incandescent light produces a stable—zero if not dimmed—amount



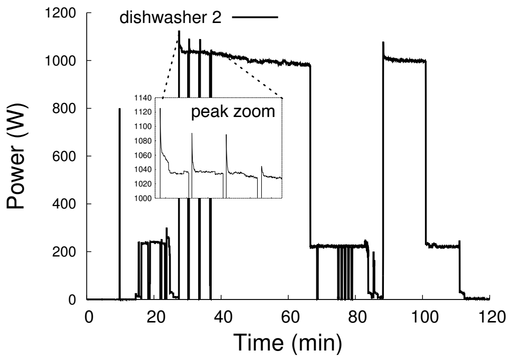
(a) Washing machine



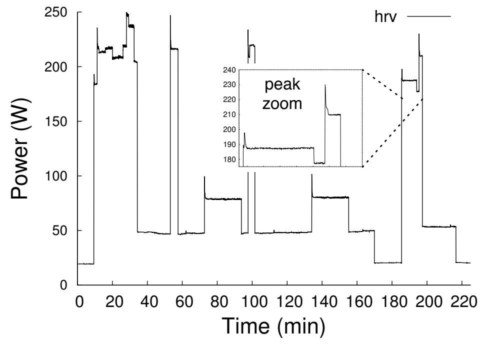
(b) Dryer



(c) Dishwasher 1



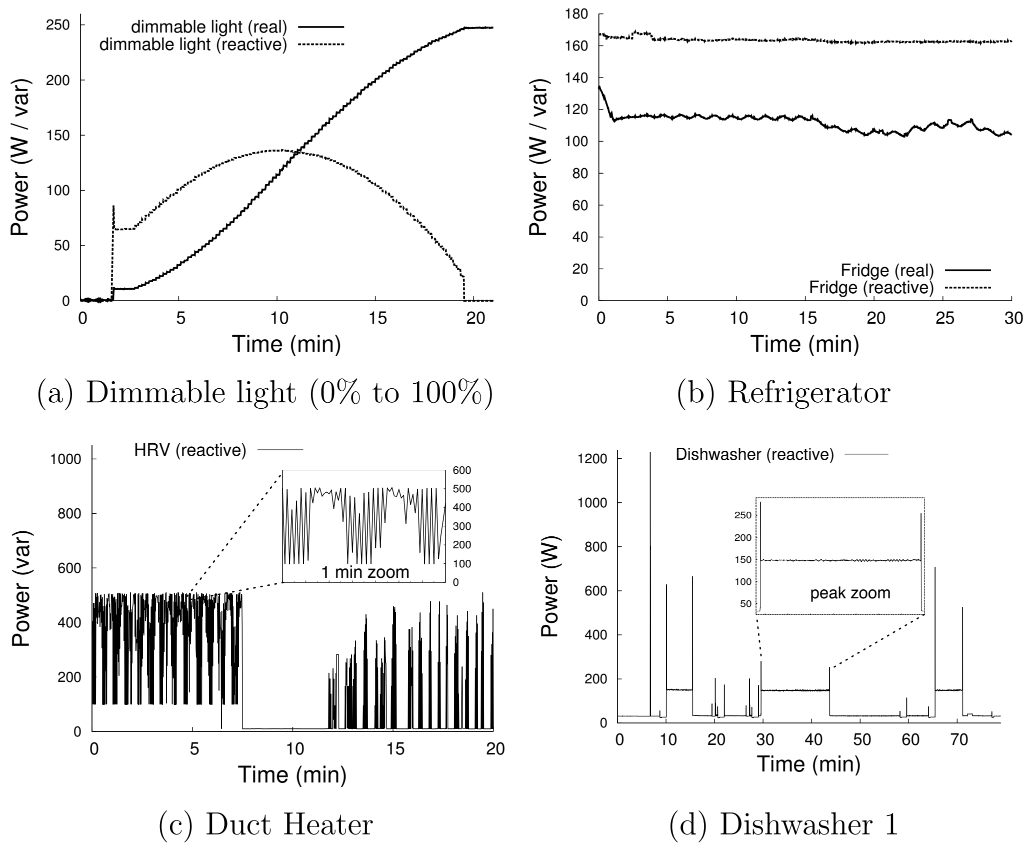
(d) Dishwasher 2



(e) HRV

Figure 4.5. Example composite loads, demonstrating combinations of the simpler loads above arranged in phases.

of reactive power when on, although the magnitude of the draw peaks at 50% dim level and decreases as the light approaches either 0% or 100% dim level. Likewise, an inductive load like the refrigerator in Figure 4.6(b) exhibits a spike followed by a flat reactive draw; a non-linear load like the duct heater in Figure 4.6(c) has a rapidly



(a) Dimmable light (0% to 100%) (b) Refrigerator (c) Duct Heater (d) Dishwasher 1

Figure 4.6. Reactive power demonstrates the same types of patterns as real power and can help in identifying different types of electrical loads.

varying power usage; and composite load like dishwasher 1 in Figure 4.6(d) operates a sequence of simpler internal loads. In each case, the pattern of a load's reactive power usage follows its pattern of real power usage.

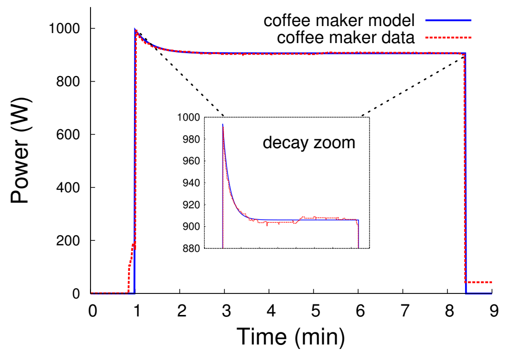
Observation 5: While the magnitude of reactive power differs from real power, a load's pattern of reactive power consumption is qualitatively similar to its real power consumption.

To summarize, we observe that *nearly every common household electric load is a composition of one or more of the small number of resistive, inductive, and non-linear loads described above*, with heating elements and AC motors consuming the majority of electricity in homes. Further, each type of elemental load exhibits similar characteristics when active: heating elements have a stable power usage or one that decays slowly over time, AC motors have a spike in power on startup and then vary their power usage smoothly over time, while SMPSs exhibit rapid and significant power variations. As we discuss in the next section, the presence of only a few elemental load types in homes simplifies model design, enabling us to accurately capture their behavior using a few basic types of models.

4.3 Modeling Electric Loads

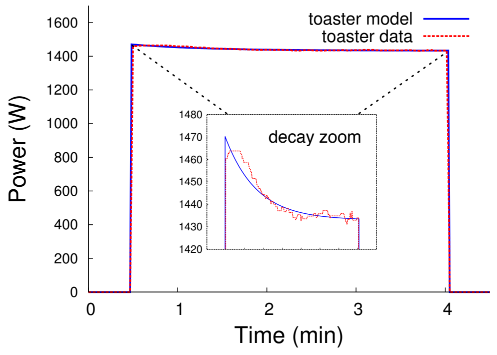
Based on our empirical observations from the previous section, we develop models to capture key characteristics of each load type. We first present four basic model types—*on-off*, *on-off growth/decay*, *stable min-max*, and *random range*— to describe simple loads, and then use these models as building blocks to form compound *cyclic* and *composite* models that describe more complex loads. Ideal models describe i) *how much* real and reactive power a load uses when active, ii) *how long* a load is active, and iii) *when* a load is active. However, in many cases, users manually control loads, such that *when* a load is active and for *how long* is non-deterministic. For example, a user may run a microwave any time for either ten seconds or ten minutes. For these loads, we assume a random variable captures this non-determinism, and focus our efforts, instead, on modeling *how* each load behaves when active.

Given each model type, we employ an empirical methodology to construct accurate load-specific models: we leverage our observations of the load's power usage as a training set, and employ curve-fitting methods to map one of the model types onto the time-series data. If the best model type is not clearly evident *a priori*, we fit multiple models and then choose the one that yields the best fit. As described below, depending on the model type, we may employ simple regression or more complex curve-fitting methods, such as LMA [41], to construct a load-specific model for a given model type. As discussed in Section 4.2, reactive power for loads exhibits



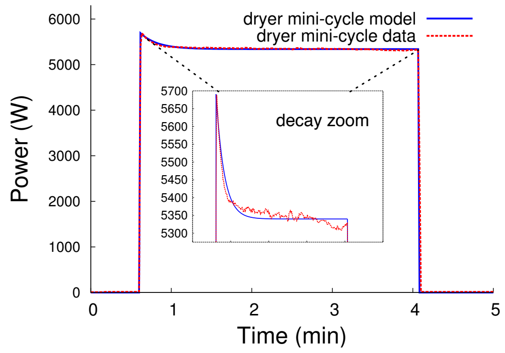
$(p_{active}, p_{peak}, \lambda) = (905, 990, 0.045)$

(a) Coffee maker



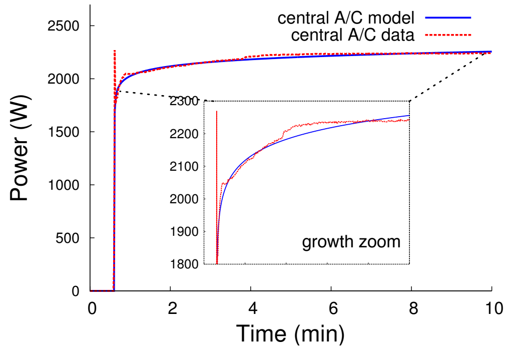
$(p_{active}, p_{peak}, \lambda) = (1433, 1470, 0.02)$

(b) Toaster



$(p_{active}, p_{peak}, \lambda) = (5340, 5685, 0.078)$

(c) Single dryer mini-cycle



$(p_{base}, \lambda) = (1800, 72)$

(d) Central A/C

Figure 4.7. An on-off growth/decay model closely matches the average power usage measured each second for a variety of resistive and inductive loads.

similar behavior as real power, and thus constructing a model of a load's reactive power consumption uses the same methodology as above.

4.3.1 Basic Model Types

On-off Model. As discussed previously, prior work often relies on simple on-off load models. An on-off model includes two states—an *on* state that draws some fixed power p_{active} and an *off* state that draws zero, or some minimal amount of, power p_{off}. Conventional, non-dimmable incandescent lights are the canonical example of an on-off load. Dimmable lights also conform to on-off models, although p_{active} depends on the dim level. As shown in Figure 4.6(a), a $N\%$ dim level yields a proportionate reduction in real power usage. In addition, while real power is a simple linear function of dim level, reactive power is a quadratic function that peaks at 50% dim level. Constructing an on-off model is simple—we use regression to determine appropriate values p_{active} and p_{off}. In particular, we partition the time series of load power usage into two mutually exclusive time-series, with data for the on and off periods, to determine the best values of p_{active} and p_{off}.

On-off Growth/Decay Model. An on-off growth/decay model is a variant of the on-off model that accounts for an initial power surge when a load starts, followed by a smooth increase or decrease in power usage over time. As discussed in Section 4.2, AC motors are the most common example of a load exhibiting this behavior, e.g., refrigerator, central A/C, vacuum. Resistive loads with high-power heating elements, such as the toaster or coffee maker, also conform to an on-off growth/decay model, although the surge and the decay in these devices is far less prominent than in AC motors. We characterize on-off decay models using four parameters: p_{active}, p_{off}, p_{peak}, and λ. The first two parameters are the same as in on-off models, while p_{peak} represents the level of inrush current when a device starts up and λ represents the rate of growth or decay to the stable p_{active} power level. We model decay using an exponential function as follows, where t_{active} is the length of the active interval.

$$p(t) = \begin{cases} p_{active} + (p_{peak} - p_{active})e^{-\lambda t}, & 0 \leq t < t_{active} \\ p_{off}, & t \geq t_{active} \end{cases}$$

Similarly, we model on-off growth as a logarithmic function (i.e., the inverse of the exponential function) using starting power level p_{base} and growth parameter λ:

$$p(t) = \begin{cases} p_{base} + \lambda \ln t, & 0 < t < t_{active} \\ p_{off}, & t \geq t_{active} \end{cases}$$

We can optionally augment the growth model with an additional parameter p_{ceil} to prevent unbounded growth that simply caps the maximum output of the model. In the growth model, the surge current must also be modeled separately (such as in the central A/C shown in Figure 4.7d). Here, we can simply add a parameter p_{spike} specifying the power at $t = 0$.

As with the on-off models above, the length of the active interval for on-off growth/decay models is often not known *a priori* since it may depend on user behavior. However, we have observed that in many cases users repeatedly operate devices in the same way, e.g., a toaster that toasts a bagel every morning. In many cases, the device determines t_{active} automatically, e.g., the compressor for a refrigerator or freezer may turn on for an average of 20 minutes in each cycle. In these cases, we incorporate the mean value of t_{active} into the model. Constructing an on-off growth/decay model requires fitting an exponentially decaying (or logarithmically growing) function onto the time-series data, in addition to determining p_{peak}, p_{active}, and p_{off}. We employ the LMA algorithm [41] to numerically find the exponential or logarithmic function that best fits the data, i.e., based on a least-squares nonlinear fit.

Figure 4.7(a) shows the specific on-off decay model for a coffee maker in parallel with its real power data. The figure demonstrates that the exponential decay is a highly accurate approximation of the coffee maker's power usage. In this case, $p_{active} = 905$, $p_{peak} = 990$, $p_{off} = 0$, and $\lambda = 0.045$. Likewise, Figures 4.7(b) and (c) show on-off decay models and real power data for a toaster and a portion of a dryer cycle. Finally, Figure 4.7(d) shows how well an on-off growth model fits the real power data for the central A/C from Figure 4.3(c). For comparison, we also fitted the lowest-error on-off model for each of the four representative device cycles pictured in Figure 4.7. We then calculated the root mean square error (RMSE) for both the on-off and on-off decay models for (a) each load's duration, and (b) its first 30 seconds of activity (including the 'on' event). As seen in Figure 4.8, the growth/decay model

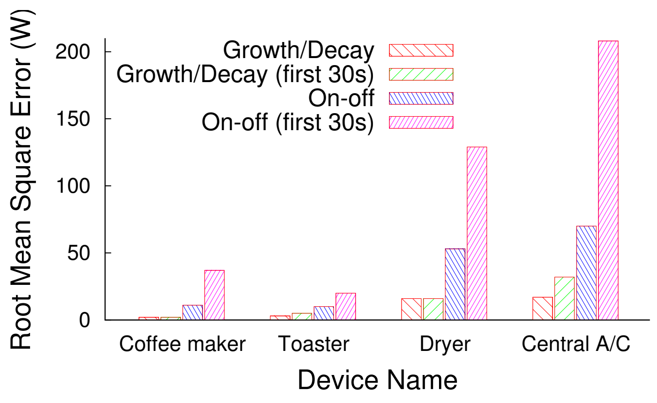
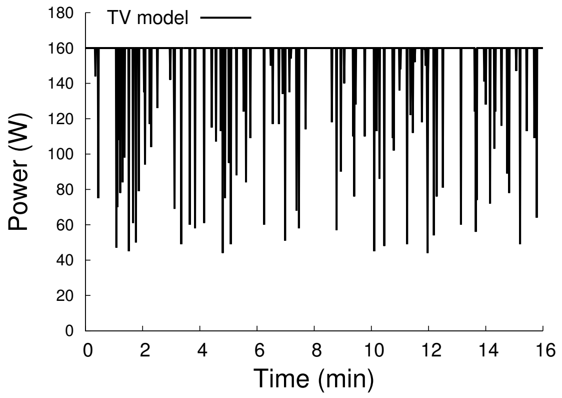


Figure 4.8. On-off growth/decay models are more accurate than on-off models.

decreases the error in the on-off model by as much as 8X, particularly in the first 30 seconds where the on-off model is unable to capture the rapid decay behavior.

Stable Min-Max Model. While on-off and on-off decay models accurately capture the behavior of resistive and inductive loads, they are inadequate for modeling non-linear loads. As seen in Section 4.2, many non-linear loads maintain a stable maximum or minimum power draw when active, but often vary randomly and frequently from this stable state. These variations are due to the device rapidly regulating their electricity usage to "match" the current needs of the device. Our stable min-max model captures this behavior by first specifying a stable maximum or minimum power when active, denoted by p_{active}. The power usage then deviates, or "spikes," up or down from this stable value at some frequency. The magnitude of each spike is chosen uniformly at random between p_{active} and a specified maximum deviation, denoted p_{spike}. The inter-arrival times of the spikes are exponentially distributed with mean λ. Thus, the stable min-max model is specified by the choice of p_{active}, p_{spike}, and λ (as well as whether p_{active} denotes a stable minimum or a stable maximum).

Empirically constructing a load-specific stable min-max model requires determining the stable power level p_{active} and characterizing the magnitude and frequency of the power spikes. We employ simple regression to determine the stable power level



$(p_{active}, p_{spike}, \lambda) = (160, 120, 10.82)$

Figure 4.9. A stable-max model of the LCD TV from Figure 4.1.

p_{active} from the data, e.g., after filtering out the data for spikes and finding the fit for p_{active}. The mean observed duration between spikes then yields the parameter λ. Figure 4.9 shows our stable-max model for the LCD TV (from Figure 4.1) using a maximum p_{active} of 160W and a λ of 10.82, which we derive from the TV's real power usage data. Importantly, as we discuss in Section 4.4, both the model and the raw data have similar statistical properties, which simple filters can recognize by detecting when power variations are significant, frequent, and symmetric, e.g., a decrease and then immediate increase in power of similar magnitude.

Random Range. Finally, we found that some devices draw a seemingly random amount of power within a fixed range when active. This is likely due to the fact that taking average power readings each second is too coarse a frequency to capture the device's repetitive behavior. We model such loads by determining upper and lower power usage bounds, denoted by p_{max} and p_{min}. When active, our model randomly varies power within these bounds using a random walk. Note that the random range model is similar to the stable min-max model in that both employ upper and lower bounds on power usage. However, while the deviations in the stable min-max model are spikes from a stable value, those in the random range model are power variations within a range. The microwave is an example of a load that exhibits this behavior. As shown in Figure 4.4(d), when turned on, the power usage of the microwave fluctuates continuously between 1400W and 1480W.

Random range models require determining the minimum and maximum of the load's range of power usage. We determine these values by simply choosing the minimum and maximum power values observed in training data, or by deriving a distribution of power values from the data and choosing a high and low percentile of the distribution to be the minimum and maximum, p_{min} and p_{max}. We then model the variations with a random walk within the range.

4.3.2 Compound Model Types

While the models above accurately capture the behavior of simple loads, many loads, including large appliances, exhibit complex behavior from operating a variety of smaller constituent loads. We devise two types of compound models for complex loads that use the basic building blocks above.

Cyclic Model. Cyclic loads repeat one of the basic model types in a regular pattern, often driven by timers or sensors. For example, the HRV heater employs a timer that activates for 20 minutes each hour. Similarly, a refrigerator duty-cycle is based on sensing its internal temperature, which rises and falls at regular intervals and fits our model well, as shown in Figures 4.3(a) and (b). A cyclic model augments a basic model by specifying the length of the active and inactive period, t_{active} and $t_{inactive}$, each cycle. Constructing cyclic models is straightforward, since it only requires extracting the duration of the active and inactive periods from the empirical data. We currently use the mean of the active and inactive periods from the time-series observations to model t_{active} and $t_{inactive}$.

Composite Model. Composite loads exhibit characteristics of multiple basic model types either in sequence or parallel. Example composite loads include dryers, washing machines, and dishwashers, as shown in Figure 4.5. *Sequential composite* *loads* operate a set of basic load types in sequence; we model them as simple piecewise functions that encode the sequence of basic load models, including how long each load operates. For instance, a model for a dishwasher is a sequence of stages: modeled as the operation of the motor (wash stage), pump (drain stage), motor (rinse stage), pump (drain stage) and heater (dry stage), where each individual stage uses an inductive or resistive load. Some loads also exhibit characteristics of two or more basic models in *parallel* if two basic loads operate simultaneously. For example, a refrigerator may simultaneously activate both a compressor and an interior light. We model *parallel composite loads* by summing the power usage for two or more of the basic model types. Finally, composite loads may also be cyclic, referred to as *cyclic composite loads*, which repeat a pattern of individual model types at regular intervals. Our methodology permits flexible compositions of sequential, parallel or cyclic loads.

Constructing load-specific composite models is more complex and requires additional manual inputs. For example, constructing a *sequential composite* model requires manually partitioning and isolating load time-series data into individual sequences that reflect the activation of the various load components. Each individual component of the composite load is modeled using a basic model type. The composite model is then simply a concatenation of these piecewise models in sequence (the duration of each component may be specified in the model or left as a variable).

As an example, Figure 4.10 shows an extended operating cycle of the washing machine with the annotations for different basic load model types in the sequence. We represent these models as large piecewise functions of the basic models describing each constituent load. In addition, many of the large appliances that have composite models also have numerous operating states. For example, the washing machine and dryer in one of our homes has over 25 different types of cycles. Ideally, a model includes a different piecewise function for each cycle type. However, in the homes we

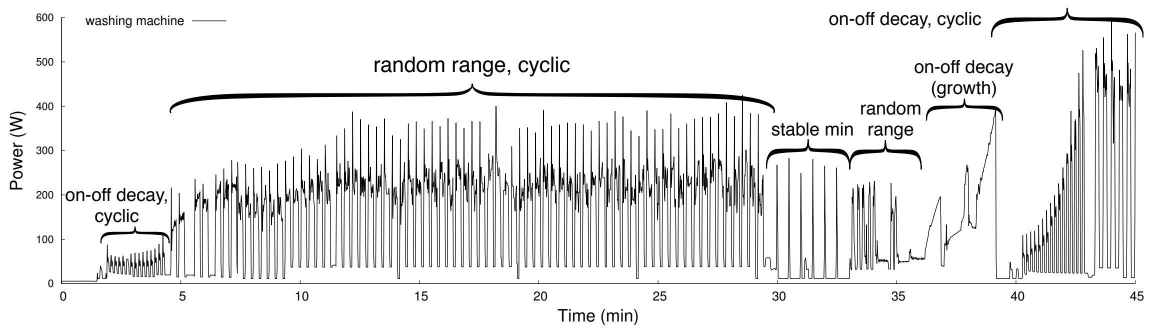


Figure 4.10. A single complete cycle of a washing machine, annotated with the model types for the operation of simpler internal loads.

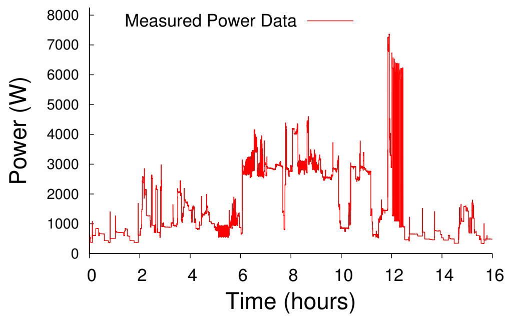
monitor we have found that most residents operate devices using only a few states—in most cases one.

Constructing parallel composite models poses additional challenges. Since the time-series data for a load captures the power usage for all components that are concurrently active, there is no straightforward general-purpose technique to extract individual models from the composite time-series data. In practice, however, extracting basic models is often possible through exogenous means. For instance, many loads permit operating individual components to isolate them for profiling, e.g., such as a running a dryer on tumble mode without any heat or using an air conditioner's fan without any cooling. After separately profiling a constituent load, such as the tumbler or fan, it is possible to operate the compressor and the fan, and then infer the compressor power usage by "filtering out" the tumbler or fan usage from the aggregate. In some devices, such as a refrigerator, it also might be possible to deploy additional sensors that monitor important events, such as a door opening that triggers lights, to filter them out. Ideally, the model of a complex composite device would be provided by the device manufacturer, as the problem of identifying the *components* of a composite device is largely orthogonal to the problem of *modeling* each component. However, using the techniques described previously, it is generally possible to identify the key components even without detailed knowledge of the internals of the device (though for many devices, substantial information on the components and operation of the device is readily available, e.g., in an owner's manual).

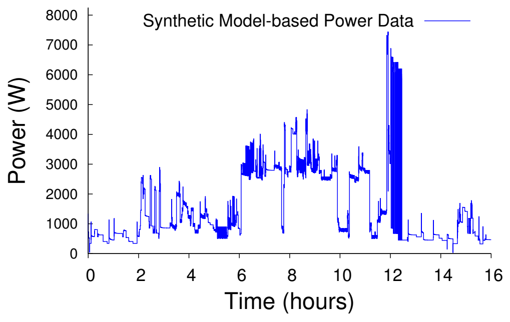
4.4 Case Study: Device-accurate Synthetic Building Data

Evaluating new techniques for analyzing electricity data requires actual building data for testing. Unfortunately, while recording a building's aggregate electricity usage is simple, requiring only a single smart meter, recording detailed aspects of the building's environment is not. For instance, evaluating the accuracy of a NILM algorithm, which disaggregates building electricity data into power data for individual loads, requires power data from *both* the entire building and each of its constituent loads. However, NILM's entire purpose is to prevent the need for recording such ground truth data at each load. As our own experience indicates, setting up even a test infrastructure for collecting ground truth data is expensive, invasive, and time-consuming, since it requires a power meter attached to each load in the home. While there are a few data sets for select buildings available for NILM researchers to use in evaluation [7, 37, 2], they typically do not instrument *every* load nor do they cover a wide range of building types or load characteristics.

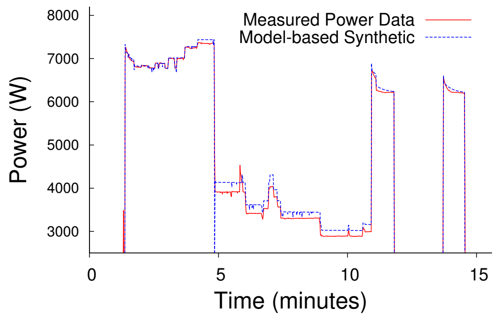
To address this problem, our first application uses our models to automatically generate device-accurate synthetic electricity data for buildings. Being device-accurate means that the synthetic trace data includes *both the synthetic aggregate time-series power data for a building, as well as time-series power data for each of the constituent loads* in the building generated using our models. While prior work targets generating synthetic traces of the power usage for entire buildings [3], we are not aware of any previous work that focuses on being device-accurate. Unlike real-world trace data collected from specific buildings, the synthetic traces generated using our models will provide researchers explicit control over the number and types of loads present in the data, enabling them to control the statistical properties of the dataset and discover



(a) Raw Power Data



(b) Synthetic Model-based Power Data



(c) Zoom-in Comparison

Figure 4.11. Aggregate home power from measured data (a) and from our corresponding device models (b), along with a zoom-in comparison over 15 minutes.

which properties have the most influence on their results. Importantly, synthetic data does not not require researchers to deploy a large number of per-load power meters.

Figure 4.11 shows an example of how our device-accurate synthetic building data compares with data collected from a real building. To generate the synthetic trace shown in Figure 4.11(b), we replace each occurrence of a given device in the ground-truth data shown in Figure 4.11(a) (i.e., each period when a device is using power) with our model of that device over the same time period. Figure 4.11(c) shows a zoomed-in comparison of the two traces over a 15 minute period to graphically illustrate their similarity. The ground-truth and model-based traces look qualitatively similar, but they also have similar statistical properties: the real data (a) has an average power of 1200W, a standard deviation of 1072W, and 5591 changes in power >15W, while the synthetic data (b) has an average power of 1165W, a standard deviation of 1073W, and 5833 changes in power >15W. Since the synthetic data is composed of data from models of individual loads, it is useful for analysis techniques that look for patterns in the aggregate usage data. By comparison, if we generate on-off models that include at most 4 power states per load (as in recent work [37]), there are only 1985 changes in power >15W, which eliminates many identifiable load-specific characteristics useful in analysis.

4.5 Summary

We address the largely unexplored problem of general-purpose, load-accurate device modeling by proposing a flexible set of models based on the low-level electrical characteristics of devices. We highlight how most devices can be represented as one or more of our basic model types and evaluate our framework by comparing to simpler on-off models used in most prior work to demonstrate our improved accuracy.

CHAPTER 5
ONLINE LOAD TRACKING

One of the most desired type of smart meter data analysis is extracting the behavior of *individual* loads given only the *aggregate* energy trace outputted by a smart meter. The modeling framework introduced in Chapter 4 enables accurate and efficient *load tracking* – i.e., extracting individual loads in an online fashion from an aggregate energy trace. Here we present PowerPlay, a system for online load tracking that is superior to traditional disaggregation techniques in scalability and fine-grained load accuracy.

5.1 Background and Motivation

Device-level energy monitoring is of widespread use both to consumers (e.g., to identify wasteful energy use in one's home) and to utilities (e.g., to perform analyses of consumer behavior under various circumstances). Timely and accurate knowledge of per-load energy usage is a prerequisite for implementing many automated energy optimization techniques [8, 12, 54]. However, gathering this information at large scales remains problematic: embedding large numbers of networked sensors in every building is either prohibitively expensive, invasive, or unreliable. An alternative approach is to analyze data from smart meters to infer individual load usage, as smart meters are being widely deployed by utilities [16]. We propose a new analysis technique, which we call *online load tracking*, that monitors the operation of individual building loads, i.e., when they turn on or off and their fine-grained energy usage, by analyzing smart meter data. In essence, “tracking” a particular load mimics having a network-connected energy meter attached to it.

Tracking loads *online*, i.e., in real time as a smart meter generates new data, is critical since many higher-level energy optimization techniques require such real-time data. For example, an automated load scheduling policy that reduces a building's peak power demand by deferring one or more background loads must know the energy usage of each background load to determine which of them to defer and for how long [8]. As another example, a recommendation engine may monitor the energy usage of a building's interactive loads to push energy-efficiency recommendations to occupants' smartphones in real-time, directing them to take an immediate action to better optimize their energy usage, e.g., such as turning off an idle coffee pot [5]. Essentially, online load tracking is useful for any application that requires attaching a power meter to a load that transmits its average power usage every pre-specified time interval in real time.

Online load tracking differs from traditional non-intrusive load monitoring (NILM) in two primary ways. First, while NILM requires disaggregating an *entire* building's energy use, tracking addresses the simpler and more tractable problem of extracting specific devices from a building that may include many unknown devices. Second, tracking focuses on *online* operation, requiring low computational overhead to provide acceptable performance. This focus differs from NILM, which primarily concerns offline analysis and does not strongly consider systems performance issues (e.g., scaling to hundreds of homes).

5.2 Tracking Overview and Approach

Our tracking system, called PowerPlay, builds on our modeling framework by detecting a small number of identifiable load features in smart meter data. For each model type, we select identifiable model features and then design efficient online methods for tracking loads by detecting one or more of these features in smart meter data. The primary components of PowerPlay are (i) a strategy for model-based feature selection, (ii) techniques for efficiently detecting features with a smart meter trace, and (iii) an implementation demonstrating the scalability of PowerPlay from single-home tracking on a low-power device up to utility-scale tracking across hundreds of individual homes. We show that PowerPlay improves per-load accuracy by >2X compared to a state-of-the-art disaggregation algorithm designed for offline analysis and easily scales to large numbers of homes on commodity hardware.

Formally, we define the problem of online tracking for load p_i as inferring its average power usage $p_i(t)$ from a home's total power usage $P(t)$ recorded by its smart meter over the period $(t - \tau, t]$. Due to its online nature, computing each $p_i(t)$ must complete within $t + \epsilon$ for some value of ϵ. Observe that tracking a load's power usage $p_i(t)$ also indirectly reveals when it turns on and off. Load tracking targets individual loads and does not attempt a full disaggregation, as is common with NILM techniques, which try to infer $p_i(t)$ for *all* n building loads, such that $\sum_{i=0}^{n} p_i(t) = P(t)$. Further, to the best of our knowledge, no prior NILM technique addresses online operation with a timing constraint.

Of course, perfectly tracking all n loads would be equivalent to a complete and accurate disaggregation. Since load tracking values system performance, as well as the accuracy of a load's inferred power readings, its goal is to *both minimize* ϵ *and maximize accuracy.* In this case, we measure accuracy based on a load's *tracking error factor* δ, which is simply the error between a load's actual and inferred power usage, normalized by its total energy usage. If $\tilde{p}_i(t)$ denotes load p_i's actual power usage at time t and $p_i(t)$ denotes its inferred power usage from load tracking at time t, then we define the tracking error factor over T intervals as:

$$\delta = \frac{\sum_{t=1}^{T} |\tilde{p}_i(t) - p_i(t)|}{\sum_{t=1}^{T} \tilde{p}_i(t)} \tag{5.1}$$

Here, the numerator is the sum of the absolute errors at each data point, and the denominator is the load's total energy usage over T. Lower values of δ are better; an error factor of zero indicates perfect tracking. While there is no upper bound on the tracking error factor, an error factor of one indicates that the reading-to-reading errors are equal to the load's energy usage. In general, a tracking error factor near one is not considered good, since simply inferring a load's energy usage to be zero at each time t results in $\delta = 1$. Note that this metric is a load-specific variant of the "total energy correctly assigned" metric from prior work [37].

Of course, the sampling resolution of the smart meter affects both the accuracy and efficiency of any online load tracking technique. We denote the meter's data resolution using the sampling time interval τ. A coarser (or longer) sampling interval "averages out" features in $P(t)$, eliminating identifiable attributes, while a finer (or shorter) interval reveals more attributes, but also more data to process, as well as more noise, e.g., due to sensing error, grid voltage variations, and non-identifiable attributes common across many loads. Our work specifically targets consumer-grade power meters, such as the TED [55], eGauge [15], and BrulTech, which commonly provide a sampling resolution of one reading per second, e.g., τ=1 second.

PowerPlay employs a model-driven approach for load tracking, which ensures accuracy and computational efficiency by decomposing tracking into multiple distinct subproblems. Note that prior work on complete load disaggregation typically conflates these subproblems. The subproblems include (i) empirically modeling a load, (ii) extracting features from the model, (iii) selecting the most identifiable features, and, finally, (iv) detecting and tracking a load based on these features. Figure 5.1 depicts the basic workflow of each subproblem, which we, in turn, outline briefly below.

1. **Empirical Modeling.** As previously detailed in Chapter 4, we first empirically model each load's energy usage based on properties of the four basic types of

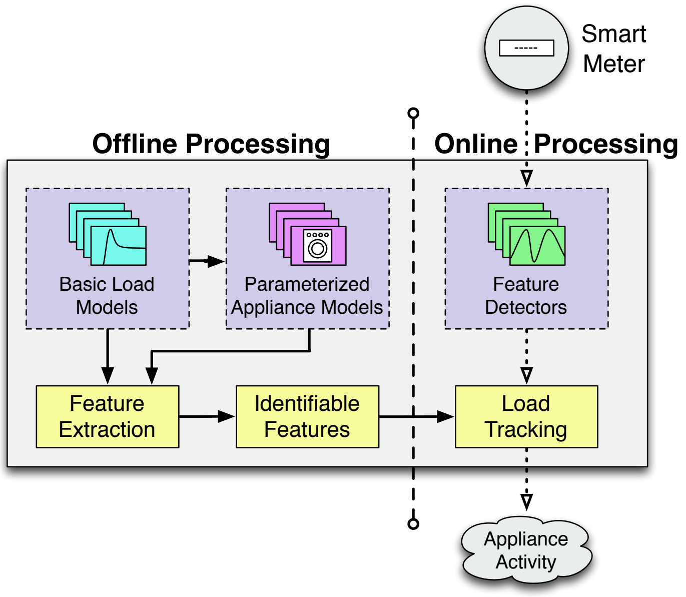


Figure 5.1. PowerPlay's approach, which uses offline modeling and feature extraction for online load tracking.

electrical loads, i.e., resistive, inductive, capacitive, and non-linear. We assume a load's model accurately describes its energy usage when on.

2. **Feature Extraction.** After empirically modeling a load, we decompose it into a set of features. Each feature captures a subset of the load's pattern of energy usage within the model: the set of features collectively represents a concise description of how the load's operation manifests itself in power data. Intuitively, a load tracking algorithm must "search" for these features within a home's aggregate smart meter data to detect the presence of the load and track it.

3. **Identifiable Feature Selection.** PowerPlay optimizes load tracking efficiency by distilling a load's full feature set into a subset of its most identifiable features. Identifiable features are a load's most prominent (and unique) features, such that a tracking algorithm need only search for these identifiable features, rather

than the full feature set, to detect and track a load with high confidence. Clearly, the smaller the set of identifiable features, the more efficient online detection.

4. **Online Load Tracking.** The final step is to design a tracking algorithm that detects a load's identifiable features in the smart meter data in an online fashion.

The first three steps above, namely empirical modeling, feature extraction, and identifiable feature selection, are one-time tasks performed offline, while PowerPlay's final detection and tracking step is continuous and online.

PowerPlay's model-based, feature-driven tracking differs from low-level time-series matching [31]. In essence, the time-series approach takes either a trace or model of a load's raw power usage when on and "matches" it against a recent window of time-series data from a smart meter to determine whether it is "embedded" in the data. Matching typically involves computing a time-series distance function, such as Euclidean distance or Dynamic Time Warping [33], between the load's raw power usage and the most recent set of smart meter readings of equal size; a match then occurs when the distance is less than a pre-defined threshold. Low-level time-series matching is more expensive and less robust than using higher-level features for load tracking.

We illustrate our approach using a face recognition analogy from computer vision. Given the image of a face, the recognition problem is to find the same face in a large library of images. One possible approach is to represent each image as a collection of pixels and attempt to find the image with the most similar pixels. Here, pixel-by-pixel matching is analogous to matching a set of points from a load's time-series power data against a building's aggregate time-series data. However, the process is error-prone, since the face in each library image may be in a different orientation or lighting, or have different glasses, jewelry, hairstyle, etc. Similarly, a load's time-series may be long and complex, causing it overlap the operation of many other load's in the building that obscure its presence.

An alternative approach is to extract the face's key features, e.g., brown hair, mustache, etc., and attempt to find faces with similar features. Matching against features is a higher-level operation than matching pixels, and attempts to capture the inherent characteristic of the problem domain (here, facial features) to perform more accurate and efficient detection. In addition, one could either decompose the face into a full set of facial features and try to match every feature, or select only a subset of prominent facial features, e.g., red hair, freckles, and blue eyes, that allow us to find matches with high confidence, but at a lower cost. PowerPlay exploits the same intuition by extracting key features of a load's model and then detecting them in smart meter data.

5.3 Offline Feature Identification and Selection

We first describe the three offline steps in PowerPlay's approach, namely modeling a load, extracting a load's features, and then selecting a subset of identifiable features to track. As noted earlier, we intend these steps to be one-time operations that occur offline. In the future, we envision manufacturers profiling each load and supplying its model and features as part of its technical user manual. The information could also be crowd-sourced. For example, The Power Consumption Database, which already provides crowd-sourced information on maximum and idle power for a wide range of loads, indexed by type, manufacturer, and model number [49], could also provide a more detailed model of each load.

5.3.1 Modeling and Feature Extraction

As previously detailed in Chapter 4, electrical loads in an alternating current (AC) system fall into one of four basic types—resistive, inductive, capacitive, or non-linear—based on the phase difference between AC voltage and the load's current waveform. Loads behaves differently based on their load type: low-power resistive

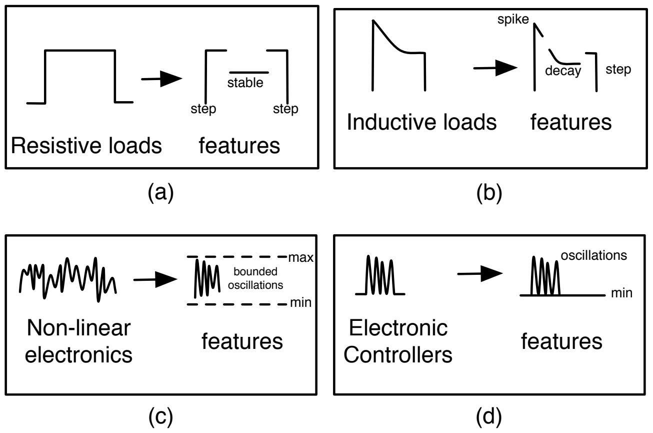


Figure 5.2. Features of the different basic electrical loads.

loads exhibit on-off behavior, inductive and high-power resistive loads exhibit spikes, decays, and growths in power, and non-linear electronics exhibit bounded or stable power oscillations. While basic loads directly exhibit one of these four behaviors, complex appliances that operate multiple internal loads, e.g., a refrigerator with a motor-based compressor and interior light bulb, exhibit a composition of these behaviors. Below, we enumerate the identifiable features from these loads that PowerPlay tracks.

Stable Power Steps. The simplest feature is a discrete change in average power from one stable value to another stable value. Most disaggregation algorithms that analyze real power data, e.g., at sampling resolutions coarser than 60Hz in the U.S., consider stable power steps as the only identifiable feature. In reality, only a few low-power resistive loads, such as incandescent lights, exhibit these simple steps and stable power usage when on. As depicted in Figure 5.2(a), we consider a power step to have three separate features: an *on* step up, an *off* step down, and a stable constant power.

Power Growth, Decay, and Spikes. Rather than exhibiting discrete power steps, many common loads experience smooth increases or decrease in power when they turn on, or abrupt and sudden spikes in power. These loads include those that operate high-power heating elements and induction motors. When turned on, the power usage of heating elements slowly decreases over time as additional heat decreases the resistance of the heating element. Similarly, the power usage of induction motors spikes at startup due to the high power required to initially start the motor from rest. As depicted in Figure 5.2(b), we consider power growths, decays, and spikes as distinct features: *spikes* capture an initial power surge, while logarithmic *growths* and exponential *decays* capture gradual increases or decreases in power.

Bounded Power Oscillations. Resistive and inductive loads are linear: they do not change the current waveform. However, many loads are non-linear: they draw current at specific times based on sophisticated electronic controllers. Thus, these loads draw a seemingly random amount of power within a fixed range when on. As depicted in Figure 5.2(c), we consider bounded power oscillations between maximum and minimum power thresholds as a distinct feature resembling a random walk between thresholds.

Stable Power Oscillations. Rather than have both an upper and lower threshold, some non-linear loads only have a single threshold, either an upper threshold or a lower threshold. As a result, these loads exhibit seemingly random power oscillations from a stable power state. This behavior stems from a load regulating its power usage to instantaneously "match" its needs. For example, a Switched Mode Power Supply (SMPSs) rapidly switches between a full-on and full-off state to conserve energy, while controllers for many heaters continuously vary their power usage to maintain a target temperature. As depicted in Figure 5.2(d), stable power oscillations are a combination of the *stable* power feature and power *spike* feature that captures frequent positive or negative random fluctuations from a stable power level.

Power Cycles. Many loads include timers that operate them periodically in a repeating pattern, e.g., a dehumidifier may include a timer that turns it on for two hours out of every four hours. Alternatively, some loads may include an environmental

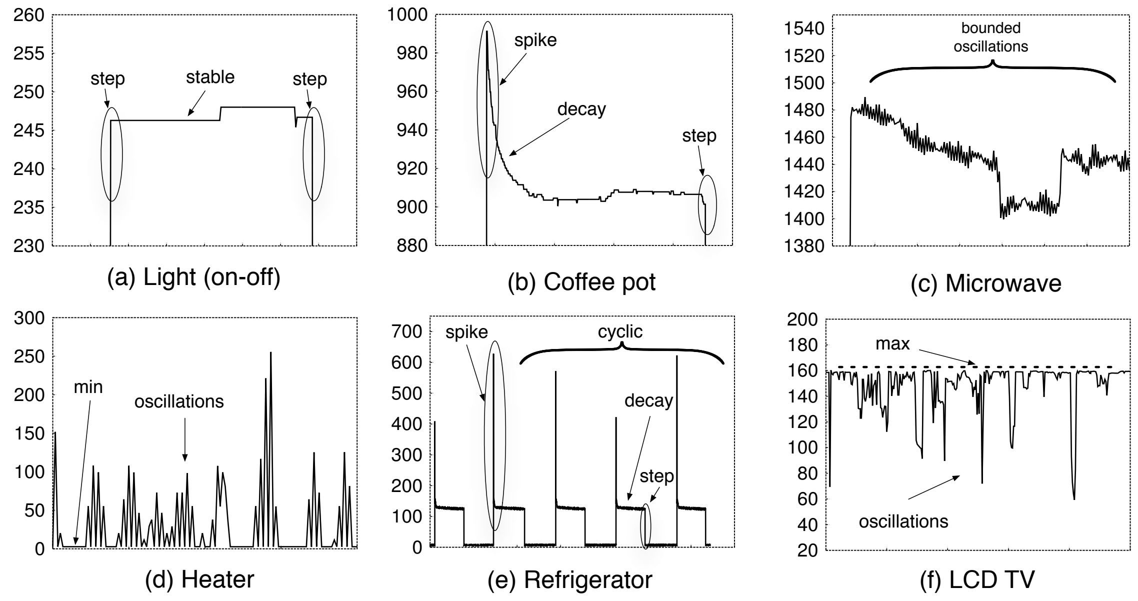


Figure 5.3. Annotated features from representative loads.

sensor that operates in a repeating pattern if the environment is stable, but slightly changes in pattern if the environment changes. A cyclic feature captures the interval and conditions at which the features repeat, and potentially their duration, e.g., the length of a stable power level or power oscillations. Note that most loads are interactive and do not repeat at regular intervals or operate over a specified duration.

Summary. Since essentially every electrical load is either an induction motor, heating element, non-linear electronics, or some combination thereof, the feature set above is complete: every load exhibits one or more of the above features. Since the feature set is small, we only require a small set of detection techniques to identify these features in smart meter data, as described in Section 5.4. Of course, since complex loads may (i) internally operate multiple simple loads in sequence, parallel, or both or (ii) operate in a regular pattern, they may exhibit a variety of the features above in an arbitrary order. For example, the washing machine previously shown in Figure 4.10 contains an induction motor for tumbling clothes, a heating element to heat water, and a pump to drain the water that each turn on in sequence during the wash cycle. Finally, note that the features above are parameterized for each specific load, and may differ across two loads of the same type, e.g., two A/Cs from different manufacturers may require different features and parameters. These parameters include the magnitude of any steps or spikes, the rate of growth or decay, the frequency and average size of bounded or stable power oscillations, the average period, etc. Thus, PowerPlay's offline component not only extracts the features of a load, it also determines the parameters for each feature. Figure 5.3 includes annotated features in power usage data for a variety of common loads.

5.3.2 Selecting Identifiable Features

Since basic loads only include a few features, an online load tracking algorithm can use all of their features to detect their presence. However, complex loads, such as the washing machine in Figure 4.10, may exhibit an excessively large number of features. The more features required to describe a load, the higher the cost of searching for and detecting these features in an online load tracking algorithm.

Fortunately, searching for every feature of a complex load is often not necessary for accurate detection; it is often sufficient to select a smaller subset of prominent features to uniquely identify the load with high confidence. To understand why, consider that, while the pattern of power usage for a particular load may be complex, it is also highly distinctive, including numerous unique features in a certain ordering. PowerPlay leverages this insight to only search for a smaller subset of *identifiable features* to match complex loads, which improves both efficiency and scale.

Selecting identifiable features for a load is a one-time offline task, and presents a tradeoff between accuracy and performance. A smaller set of identifiable features improves the efficiency of detection, but decreases tracking's accuracy. At present, we construct a complex load's set of identifiable features experimentally by iteratively adding the next highest magnitude features, e.g., that include the largest changes in power, to the feature set and then executing our tracking algorithm on historical data until the tracking error factor is below a pre-defined threshold, e.g., 0.10.

5.4 Online Load Tracking

In this section, we first describe PowerPlay's online tracking algorithm and then describe the various feature detection techniques the algorithm uses to detect the features from Section 5.3. The right side of Figure 5.1 depicts this process.

5.4.1 Tracking Algorithm

PowerPlay's tracking algorithm takes, as input, a set of loads to track, a set of identifiable features for each load, and a continuous stream of data from a smart meter. Feature detectors for each load operate over a moving window of data points of size W, starting from the most recent data point in the time-series of a home's power readings. The window represents the minimum time period over which a feature manifests itself. The output of the tracking algorithm acts as a set of virtual power meters providing device-level power data for each tracked load.

PowerPlay orders the list of all identifiable features across all loads into three sets, from most to least distinctive. The first set contains "noisy" features, namely, all stable and bounded power oscillation features across all loads in the tracking set. The second set contains the remaining basic features: steps, spikes, and decay/growth features across all loads. The final set contains any cycle features for loads in the tracking set. Given these ordered sets, the tracking algorithm then repeatedly executes its main loop, which applies every feature detector (from all loads) in order, as described in Section 5.4.2. Note that PowerPlay buffers any smart meter data that arrives while executing its main loop, and reads and appends it to the home's power data time-series on the loop's next iteration. The time taken to complete the main loop defines PowerPlay's online performance, i.e., the minimum ϵ it can support. For example, if the main loop takes 30 seconds to complete, then the tracking algorithm can only output each load's inferred power usage every 30 seconds. The exact value of ϵ depends on available hardware resources, as well as the number of virtual power meters to simulate – i.e., twice as many tracked loads will increase ϵ by roughly 2X.

PowerPlay first detects the "noisy" features, i.e., those that contain significant power fluctuations. These features are detected, labeled, and filtered from the home's power data as described in Section 5.4.2. Detection and filtering of "noisy" features first enables PowerPlay to more easily and accurately detect the remaining features, as the residual filtered data has less noise after filtering. After filtering, PowerPlay applies the remaining basic feature detectors (e.g., spikes, growth/decays, and steps) to identify and label those features in the data. Finally, PowerPlay runs the cycle feature detector over the list of labeled features to identify repeating patterns of features – the cycle feature detector is unique in that its input is a set of labeled features rather than raw time-series data, and as such is run last.

For each desired virtual power meter (i.e., load in the tracking set), PowerPlay then examines the list of labeled, but unassigned, features found in the recent past (over a window W). If the identifiable features of the load are found in the window, it assigns these features to the load and declares a load match. Upon assigning features to a load, PowerPlay removes them from the list of unassigned features. For composite loads, the set of features (over window W) may need to occur in a certain order (or within a certain time interval) to infer a load's presence. Finally, whenever PowerPlay detects a load based on its features, it updates the load's inferred power usage $p_i(t)$ using the filtered feature data and the load's model, which captures the load's full power usage behavior. Algorithm 1 shows pseudo-code for our load tracking algorithm.

Algorithm 1: PowerPlay's Load Tracking Algorithm

```
Inputs:
 list of loads to track;
 set of identifiable features per load;
Preprocessing:
 group all features based on “noise”;
  1st group: stable min-max and bound oscillations;
  2nd group: spikes, growth/decays, and steps;
  3rd group: cycles;
while true do
    Read in new, unprocessed smart meter data from buffer;
    Append new data to existing (filtered) power data time-series;
    Execute every stable min-max and bounded oscillation feature detector on
    filtered data;
    if Match then
        Identify and label feature;
        Filter feature from power data time-series;
    end
    Execute each spike, growth/decay, and step detector on filtered data;
    if Match then
        Identify and label feature;
        Filter feature from power data time-series;
    end
    Execute each cycle detector across labeled features;
    if Match then
        Identify and label cycle;
    end
    for each load in the tracking set do
        if features present (in specified order) then
            Identify load's presence;
            Reconstruct load's inferred power time-series from:
             filtered load features and full model;
        end
    end
end
```

5.4.2 Feature Detection

PowerPlay's tracking algorithm relies on individual feature detectors to identify the features described in Section 5.3, including power steps, spikes, growth/decay, bounded oscillations, and stable min-max oscillations. We detail each of these feature detectors below. Note that, as stated above, each feature i operates on prior data over a window size W_i, starting from the most recent data, where the value of W_i is specific to feature i. As with other similar types of analyses, feature detectors transform raw power readings into a series of changes in power, which we call *power deltas* or just deltas, e.g., +50W, -30W, +25W, etc., before processing them. PowerPlay associates each power delta, e.g., a +100W, with one and only one feature from a single load, removing it from further consideration by other feature detectors.

Stable Oscillation Detector. This detector examines data for frequent power oscillations from a stable minimum or maximum power level, such that for every negative power delta there is a corresponding positive power delta (of near equal magnitude) in the near future. More formally, it identifies a stable power oscillation feature by scanning a recent window of data, while maintaining a stable power level p, which it updates only if power deviates from p by at least T watts for at least D seconds. The parameters T and D are specific to a particular device that exhibits this feature. We consider changes in power that update p as background activity, which we exclude from the stable power oscillation feature, while we label any other oscillations within the window as part of the feature. Finally, we cluster nearby groups of labeled points to indicate the time range the feature was present.

To filter the feature from the raw data, we remove from the data any oscillations that do not result in an update to p, and then use them to reconstruct the feature's second-to-second energy usage due to its stable oscillation behavior, as detailed in Algorithm 2 and illustrated in Figure 5.4. In determining the D parameter for each load, the goal is to set it long enough to ensure changes in power are not random oscillations due to some other load, but short enough to prevent filtering short-lived loads. For T, the goal is to select a value large enough to capture the expected oscillations without attributing the power usage of unrelated background loads to the feature.

Bounded Oscillation Detector. The bounded oscillation detector examines data for groups of deltas within a certain range that reverse themselves—change

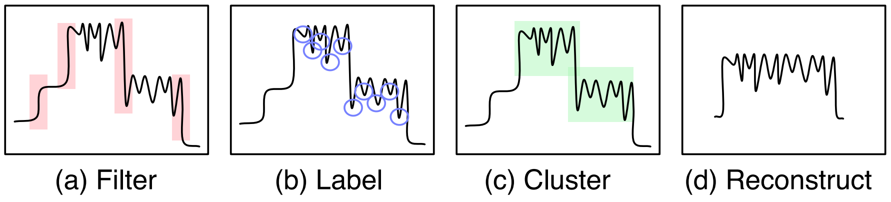


Figure 5.4. Detection of a stable oscillation feature.

from positive to negative—frequently within a given minimum window size (e.g., 60 seconds). In particular, the detector looks for a minimum proportion of reversals within the window (e.g., 50%), extending the window size until the minimum proportion is not met or several seconds have passed without a reversal (i.e., power use has stabilized, indicating the device is off). Within the resulting window, power deltas exceeding the bounded power range are filtered out, as these changes are presumably caused by other devices. This filtering is analogous to the the first step of the stable oscillation detector shown in Figure 5.4.

As an example, we might parameterize a bounded oscillation feature for a particular microwave by dictating that at least 50% of reversals over its time window are within a 30W range. Thus, over the initial small window, e.g., 15s, there must be at least 8 reversals to detect the feature, at which point the detector extends the window until (i) the minimum reversal percentage no longer holds, or (ii) a short period passes, e.g., 10s, without any reversals. This approach serves to extend the window as long as necessary without overly lengthening the window for long-running loads. To extract the resulting feature, we pair active windows of reversals with matching on and off power steps of the approximate expected size for the feature (e.g., 1000W for a particular microwave), as illustrated in Figure 5.5.

Growth/Decay Detector. To detect a decay or growth feature, we identify positive steps near a feature's expected magnitude, representing possible 'on' events. Since the expected decay or growth rate specifies a maximum per-second negative

Algorithm 2: Stable Oscillation Detector

```
Inputs:
  data window $W = \{w_0, w_1, \ldots, w_n\}$;
  threshold wattage $T$;
  threshold duration $D$;
$p \leftarrow w_0$;
$sinceStable \leftarrow 0$;
for $i$ from 0 to $n$ do
    if $|w_i - p| \geq T$ then
        $sinceStable \leftarrow sinceStable + 1$;
        if $sinceStable = D$ then
            $p \leftarrow w_{i-D}$;
            label($w_{i-D}$, $BACKGROUND$);
            $i \leftarrow i - D$;
        end
        else
            label($w_i$, $OSCILLATE$);
        end
    end
    else
        $sinceStable \leftarrow 0$;
    end
end
for each $w_i$ in each $OSCILLATE$ cluster $C = \{w_m, w_{m+1}, \ldots, w_{m+k}\}$ do
    if $w_i$ labeled $OSCILLATE$ and not $BACKGROUND$ then
        output delta $d_i = w_i - w_{i-1}$;
    end
end
```

step (for a decay) or positive step (for a growth), the detector then scans forward, discarding all changes that exceed the expected maximum. The result of this process is a filtered time-series that, assuming the data actually represents a growth or decay, should approximately fit an exponential or logarithmic curve. The detector then performs the standard Levenberg-Marquardt Algorithm (LMA) [41] to perform curve fitting against the data. If the fit fails, or the derived decay/growth parameter is far from the expected value, the detector moves on to the next possible 'on' event.

If the fit is successful, then the detector identifies the 'off' event for the device, or, equivalently, the duration of the decay/growth. To do this, the detector gradually extends the fitted curve while looking for an 'off' step of the expected magnitude,

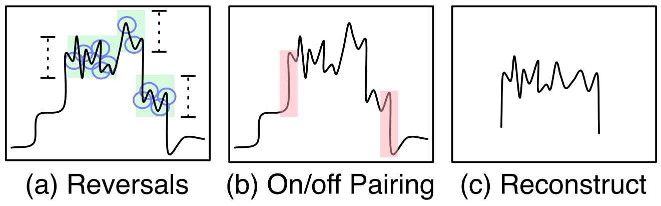


Figure 5.5. Example of bounded oscillation detector.

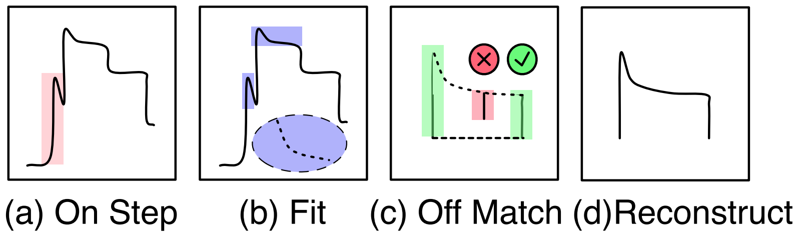


Figure 5.6. Operation of the decay/growth detector.

based on the magnitude of the 'on' step plus the cumulative growth or decay of the fitted curve, which increases with the length of the curve. The detector then chooses the 'off' step within a bounded interval most closely matching the expected value. In this case, bounding prevents a runaway search. After selecting the 'off' step, the detector is able to trivially reconstruct the entire feature, based on the identified 'on' and 'off' events and the fitted curve between them. The process of fitting and filtering a decay feature is shown in Algorithm 3 and illustrated in Figure 5.6.

Spike Detector. Power spikes manifest themselves across multiple seconds, either due to variation in a load's exact activation time, i.e., when it activates within the one-second sampling interval, or due to a short ramp-up period, which is especially prevalent in high-wattage loads. Thus, the spike detector collapses consecutive power steps in the same direction, e.g., up or down, into a single aggregate power step. Once collapsed, we identify spikes by a large positive step, followed *immediately* by a smaller, but still significant, negative step (currently, at least 30% of the positive step). Importantly, the spike detector separates the spike itself from its load's stan-

Algorithm 3: Growth/Decay Detector

```
1  Inputs:
2    data window $W = \{w_0, w_1, \ldots, w_n\}$;
3    expected feature 'on' size $s$;
4    decay/growth parameter $\lambda$;
5  for each delta $d_i = w_i - w_{i-1}$, where $d_i \approx s$ do
6      $fitData \leftarrow (d_i)$;
7      for each delta $d_j = d_{i+1}, d_{i+2}, \ldots$ do
8          if $d_j < maxChange(\lambda)$ then
9              append($fitData$, $d_j$);
10         end
11         else
12             append($fitData$, lastOf($fitData$));
13         end
14     end
15     $\lambda_{fit} \leftarrow LMA\_Fit(fitData)$;
16     if $\lambda_{fit} \approx \lambda$ then
17         for fit length $f$ from 0 to $n$ do
18             if $d_{i+f} \approx -s + totalChange(\lambda, f)$ then
19                 output ($d_i$, $f$, $d_{i+f}$);
20                 exit;
21             end
22         end
23     end
24 end
```

dard power step feature. For example, PowerPlay considers the series of changes in power [0, 0, +500, -400, 0, 0] both a +100W power step feature with a 500W power spike. Although the naïve step-only approach would output a +500W step and a -400W step, the spike detector recognizes that this time-series most likely represents a 100W inductive load, such as a 100W refrigerator. Since the magnitude of a spike is highly influenced by when a load turns on within the sampling interval, we represent the spike as a binary flag associated with the regular power step feature, e.g., the +100W step in our refrigerator example.

Step Detector. While power steps are the simplest feature, we have found that a trivial approach to identifying them—by detecting second-to-second power deltas of a certain magnitude in the data stream—is often inaccurate for the same reason

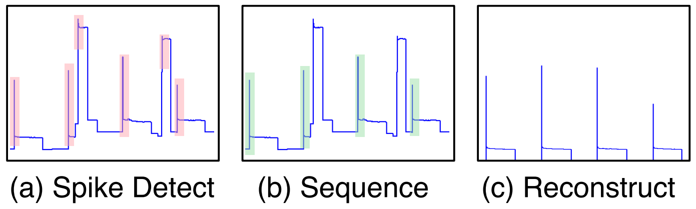


Figure 5.7. Operation of the cycle detector.

as above, e.g., loads turn on at different times within the sampling interval resulting in wide variance in the step feature's recorded power deltas. Thus, similar to the spike detector above, we collapse multi-second power deltas in the same direction into a single aggregate delta. Once collapsed, the detector simply compares a step's magnitude to a specific parameterized power step feature. Note that we exclude power deltas previously assigned to other features—-most notably the frequent power variations observed in the stable min-max and bounded oscillation features—since they would generate large numbers of spurious power steps of potentially the same magnitude as an actual load.

Cycle Detector. Cyclic loads exhibit one or more features in a regular pattern. Unlike the detectors above, the cycle detector operates on a series of labeled features (from the detectors above), and then i) identifies each potential cyclic feature from the data and ii) chooses a sequence of the features that most closely matches the cycle's expected period length. Figure 5.7 illustrates the process, where the cyclic feature is a spike.

To determine the best sequence of cyclic features of a particular type, we chose an arbitrary cyclic feature of the type at time t_1, then the next one closest to time $t_2 = t_1 + period$, and so on for $t_k = t_{k-1} + period$. To account for features missed by its particular feature detector, we may also match t_k to $t_k = t_{k-1} + 2 * period$. The 'error' of the resulting sequence of t_k is computed as $\sum_k |t_k - t_{k-1} - period|$, i.e., the amount the sequence differs from the expected period. This error is computed for all sequences starting from each possible t_1, and the detector selects as the predicted cycle the sequence with the lowest total error.

For example, consider a refrigerator with a 30 minute period and a magnitude range between 80W and 120W for its spikes at startup. Now suppose the detector extracts all spikes (due to the refrigerator's compressor) from the data, and of those spikes, each one with a step between 80W and 120W occur at times [0m, 20m, 30m, 55m]. In this case, the detector labels events at 0m, 30m, and 55m as the 'on' events of the refrigerator, while excluding the the event at 20m, as it is does not match the expected period. While this is a brute-force approach, the relatively small number of cyclic loads, ensures the process is not computationally expensive.

After determining the sequence of cycle 'on' events, we filter and reconstruct the feature's energy usage by filling in its corresponding load's model starting from each 'on' event, as shown in the final step of Figure 5.7.

5.5 Implementation

We implement PowerPlay's feature detectors and tracking algorithm as a library in Perl. The input to PowerPlay's tracking algorithm is a continuous stream of new smart meter data, which PowerPlay buffers while executing its main loop. Thus, if each iteration of the main loop takes ϵ time, then the next iteration will consider the set of data points that arrive and are buffered over the previous ϵ. The tracking algorithm also has, as input, the set of loads to detect and the corresponding set of identifiable features (parameterized separately for each load) extracted offline. The algorithm then outputs, for each load, its inferred per-second power usage over ϵ for each iteration of the main loop, resulting in a separate time-series of power data for each load in the tracking set.

We deploy PowerPlay in one of the real smart homes described in Chapter 3. The extensive instrumentation present in our deployment is necessary to compare our results (computed from the home's aggregate power data) with the ground truth power data from each individual load.

Notably, we must manually model each load we track and extract its important features ourselves in order to use PowerPlay. However, our hope is that by demonstrating the usefulness of our models in analysis, we will motivate manufacturers to use our methodology to derive models and extract features as part of a load's design and publicly release them.

5.6 Evaluation

We evaluate the accuracy and efficiency of PowerPlay's online load tracking algorithm in our home deployment. We first measure the computational overhead of load tracking to quantify PowerPlay's *efficiency*, which enables it to either track loads on low-power embedded platforms, i.e., within a home, or scale to thousands of loads (across many homes) on server platforms. We then evaluate PowerPlay's *accuracy* by quantifying the tracking error factor δ for various loads. In both cases, since there is no prior work on load tracking, we compare PowerPlay to a complete disaggregation algorithm (based on FHMMs) modified for online operation. In this case, we use the same approach as Kolter and Johnson [37] to evaluate their Reference Energy Disaggregation Dataset (REDD), which is similar to the technique by Kim et al. [34].

Since PowerPlay relies on load models computed offline, we manually model a representative set of loads in our deployment home that collectively cover each feature type. The set includes a toaster oven (steps, decays), a refrigerator and freezer (steps, spikes, cycles), a heat recovery ventilator or HRV (stable oscillations), and a dryer (bounded oscillations, cycles, steps, decays). PowerPlay then tracks these loads, i.e., infers their second-to-second power usage, in real time using per-second power data for the home (supplied by the home's smart meter). The home operates 92 distinct loads.

5.6.1 Tracking Efficiency

As Algorithm 1 indicates, PowerPlay operates online by continuously receiving power readings each second and executing its main loop to perform feature detection on the most recent window of data. Since PowerPlay stores recent data in memory, I/O overhead is negligible and efficiency is solely a result of the computational overhead of the feature detectors.

This overhead determines both (a) the tracking delay (ϵ from Section 5.2) of the system, where ϵ=1 second is perfect real-time tracking and ϵ=1 hour indicates the system delays reporting a load's power usage by one hour, and (b) the number of loads (and homes) that a platform can effectively track. Note that, since PowerPlay's main loop detects features across all loads, increasing the number of loads, ignoring parallelism, increases tracking delay across all loads. Thus, we measure the aggregate number of loads PowerPlay can track, while maintaining a low tracking delay.

We perform the following experiments on a single-core server running Ubuntu Linux (kernel version 3.2.0) with a 2.4GHz Xeon processor. Figure 5.8 plots the tracking delay (ϵ) PowerPlay supports for each of our representative loads as a function of the tracking window size. For this experiment, we set a common window size across all features and vary it, even though some features do not require large window sizes for accurate detection. From the graph, we observe that the tracking delay is modest across every load. For example, with an excessively long tracking window of 24 hours, PowerPlay completes in less than 3 seconds per load.

As expected, the more features a load exhibits, e.g., the dryer, the more computational overhead required to track it, and the longer the tracking delay. We also observe that the tracking delay effectively varies linearly with the tracking window size. As a result, shortening the window size linearly decreases the tracking delay. In practice, we select the window size specifically for each feature. For example, while cyclic features may require a few hours to manifest themselves (requiring a multi-hour

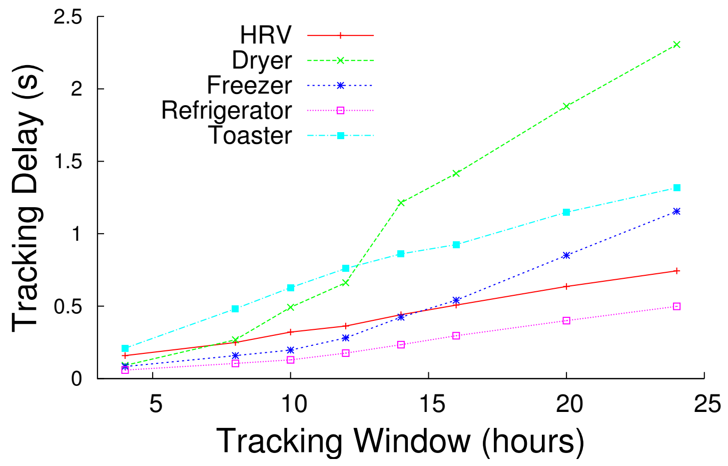


Figure 5.8. PowerPlay's tracking algorithm is efficient, with tracking delays of at most a few seconds.

window), spikes and decays are typically evident within a few seconds (requiring a window of only a few seconds). Of course, tracking most features requires significantly less than a 24-hour window size, which typically results in sub-second tracking delays.

Result: *PowerPlay is able to track multiple loads in real-time, or near real-time, on commodity servers.*

As noted above, we also compare PowerPlay's scalability with a complete disaggregation algorithm based on FHMMs. Here,we assume a server must track loads across *many* homes, not just a single home. For example, utilities might apply load tracking to large electrical loads, e.g., electric heaters and air conditioners, across thousands of homes at grid scales to estimate their demand response capacity, e.g., how much they can reduce grid demand by deferring elastic loads. We quantify both PowerPlay's performance (with 24-hour and 4-hour tracking windows) and an FHMM strawman modified for online operation. For our FHMM, we use the same parameters as Kolter and Johnson use [37].

Since disaggregation using the FHMM is exponential in the number of building power states (which is based on the number of loads and the number of power states per load), Kolter and Johnson model each load as having only four power states and disaggregate at the level of individual circuits, rather than individual loads. Our FHMM performs similarly: we model each load as having four power states and disaggregate at the level of individual circuits. Since our home has only 25 circuits, but operates 92 individual loads, our FHMM performance numbers for a complete disaggregation are conservative.

Since the FHMM approach requires a sizable amount of data, e.g., 24 hours, for complete disaggregation, it cannot operate on a small window size. As a result, our modified FHMM executes a similar main loop as PowerPlay, but always disaggregates the most recent 24 hours of data. We do not explore how to increase its efficiency by modifying its algorithm to be incremental. Our strawman online FHMM incurs an 86 second tracking delay to track the loads in Figure 5.8 for a single home. In contrast, PowerPlay imposes only a 5.6 second and 0.6 second delay for the 24-hour and 4-hour tracking windows, respectively, for the same home.

We also plot the scalability of each approach on a quad-core server running at 2.4GHz in Figure 5.9, where the number of independent homes we track is on the x-axis. Note that tracking each home is an independent process that runs in parallel. The graph demonstrates that the FHMM approach does not operate in real-time: even tracking loads in a mere 10 homes imposes a tracking delay greater than 10 minutes. PowerPlay performs much better with the same 24-hour time window, supporting roughly 100 homes with a tracking delay of 2.5 minutes. The more realistic scenario, with a smaller 4-hour time window, scales even better: PowerPlay tracks each of the five loads in 1000 homes (or 5000 total loads) with a tracking delay of only 2.5 minutes.

Result: *PowerPlay scales to support online tracking of many homes; in this case, tracking 5000 loads across 1000 homes with a tracking delay of only 2.5 minutes.*

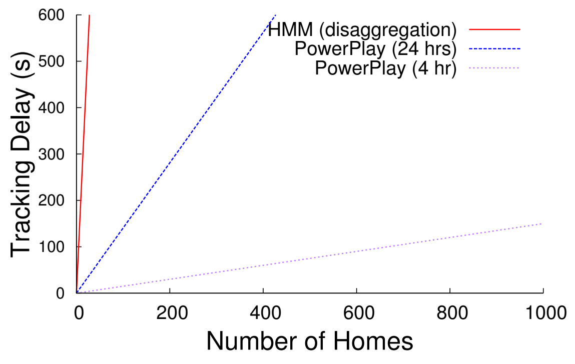


Figure 5.9. PowerPlay efficiency enables it to scale to many homes, while maintaining a low tracking delay.

Finally, we also consider PowerPlay's performance on embedded platforms that track a set of loads *within* a home. This scenario is important, since we envision PowerPlay potentially running on smart meters, themselves, or other types of commercial meters. Recent work proposes such an embedded energy monitoring and analytics platform [36]. To evaluate this case, we deploy PowerPlay on a low-power DreamPlug computer with a 1.2GHz ARM processor and 512MB memory, costing less than $100. We then tracked the same five loads as above in our deployment home. Our results show that with a 4-hour tracking window, the tracking delay was just 18 seconds, with individual load tracking times ranging from less than a second for the refrigerator to four seconds for the toaster.

Result: *PowerPlay is capable of online tracking of loads within a home on low-power embedded platforms.*

5.6.2 Tracking Accuracy

In addition to efficiency, load tracking must also be *accurate* to be useful. As before, we compare PowerPlay's accuracy in tracking multiple loads' real-time power usage with the FHMM approach, which performs a complete disaggregation. Note that FHMMs require training data to build their own internal models. Here, we take the conservative approach of training the FHMM on per-load data from the home that

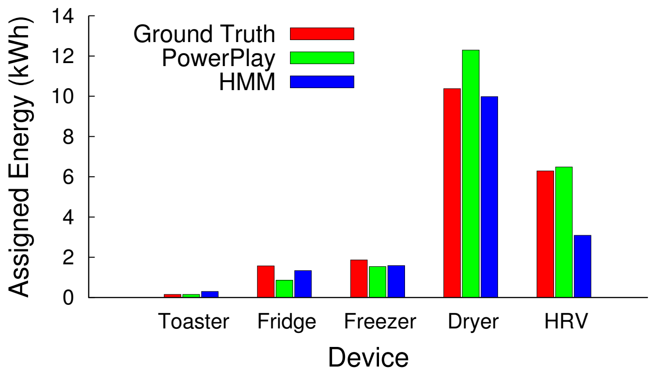


Figure 5.10. Both PowerPlay and the FHMM approach accurately assign the energy used by loads each day.

we disaggregate, which improves its accuracy. In practice, this is not often possible, since disaggregation is typically only useful in homes where such training data is not available, requiring training data from different homes.

Disaggregation often focuses on inferring a breakdown of per-load energy usage for a building over a long time period, e.g., an entire day or week. Figure 5.10 shows the actual energy usage over an entire day for five loads, as well as the inferred energy usage from PowerPlay and the FHMM disaggregation. We see that both PowerPlay and FHMM accurately predict each load's energy usage over long periods of time, although the FHMM approach is less accurate for the heat recovery ventilator due to its stable power oscillations. Even for lower-power loads, including the toaster, freezer, and refrigerator, both approaches predict energy usage near that of the load's actual energy usage. Our results are consistent with prior work on the FHMM approach, which performs as well, or better, than other prior approaches to disaggregation [34, 37].

Result: *The accuracy of PowerPlay's inferred energy usage for loads in the tracking set over long periods is comparable to that of complete disaggregation via a FHMM.*

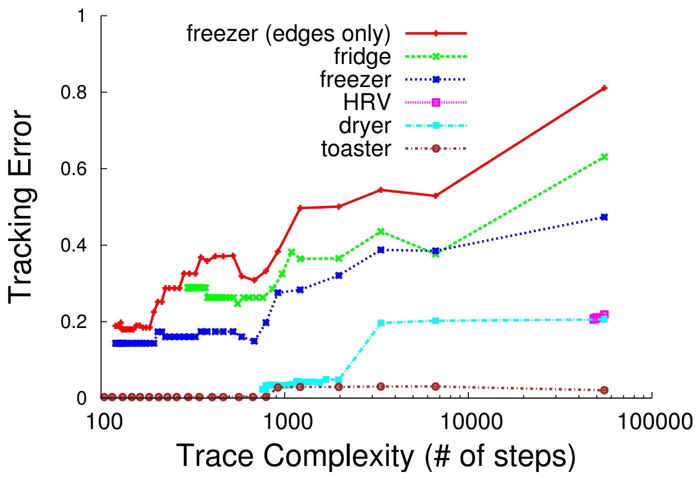
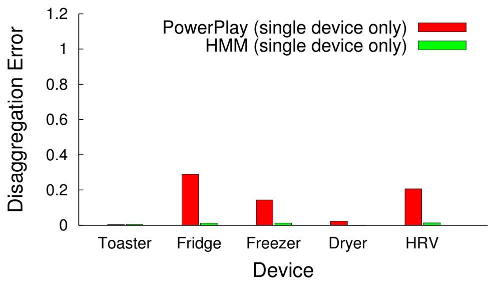


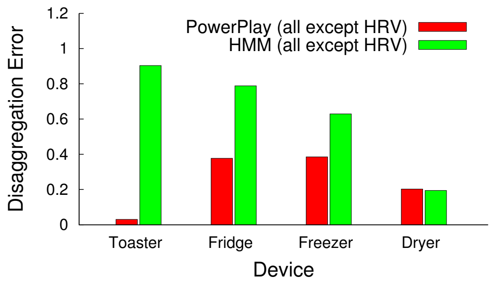
Figure 5.11. PowerPlay error factors when scaling up to highly noisy and complex smart meter data.

Unfortunately, inferring energy usage over a long period is not appropriate for online operation, and does not take into account *when* a load uses energy. We use the tracking error factor δ from Section 5.2 to quantify per-load accuracy over time.

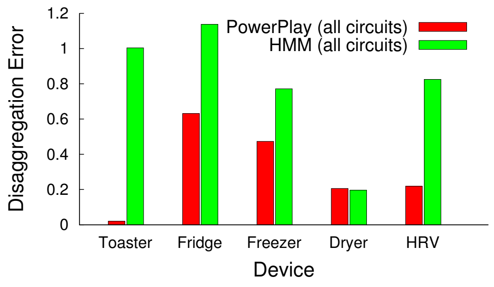
In Figure 5.11, we first quantify accuracy as we scale up the number of non-tracked loads in a home, since more loads result in more (and less visible) features. In this case, the x-axis is a rough measure of the data's complexity, i.e., the number of power deltas >15W. We create increasingly more complex smart meter datasets using our home deployment by adding more circuits to each dataset. For example, the far left side of the graph includes only one circuit (the one including the corresponding tracked load) and each data point to the right represents a dataset with one more additional circuit added to it. For each new circuit, we track the loads and compute the error factor per load on the new dataset. Figure 5.11 plots the results for our representative loads. Note that the x-axis is on a log scale, since a small number of loads contribute the majority of the power deltas. For comparison, we also include a second model of the freezer that only uses step features, to illustrate the effect of removing all but the most trivial features present in PowerPlay.



(a) Self Input



(b) Aggregate Data minus HRV



(c) Aggregate Data

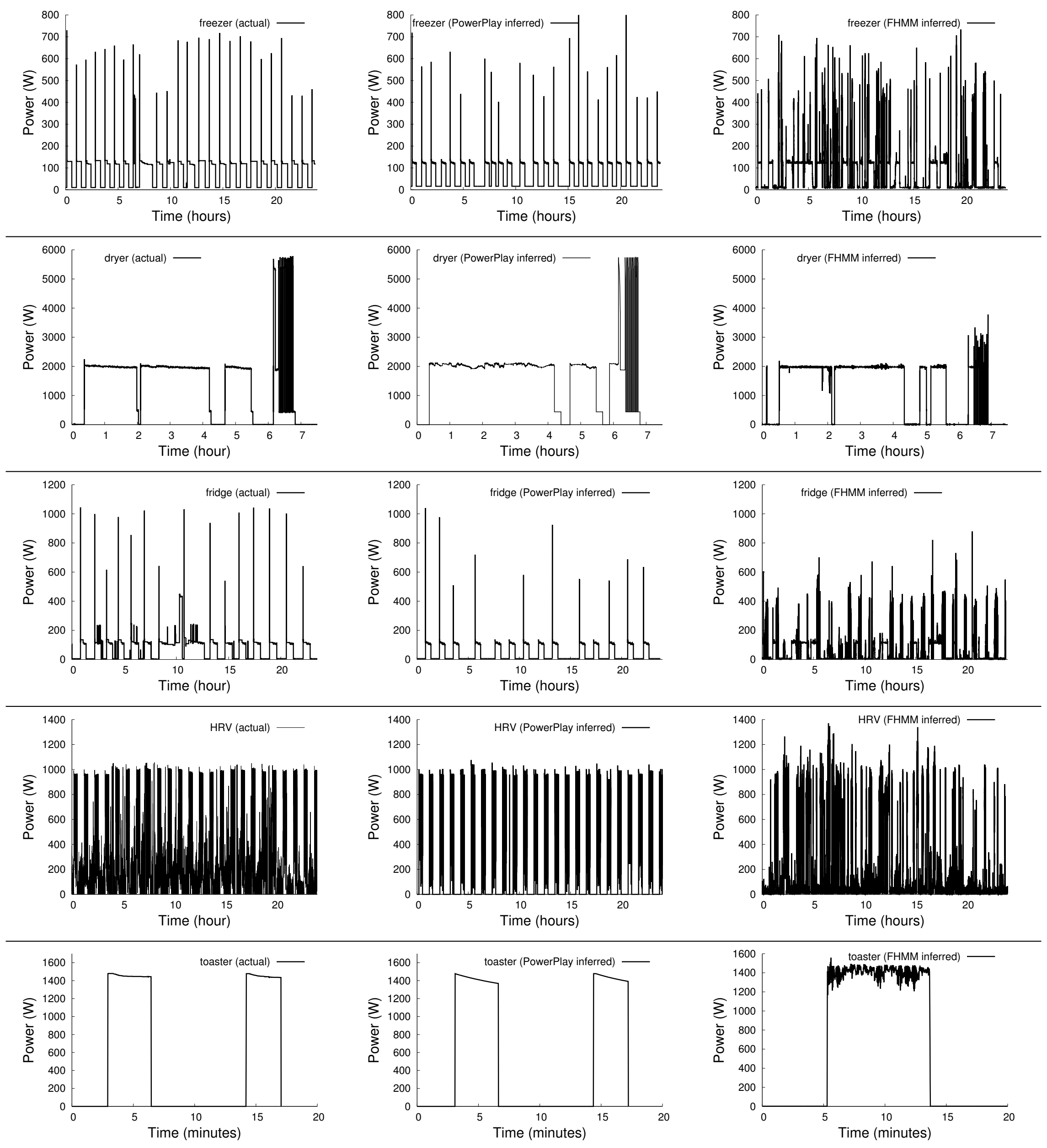
Figure 5.12. PowerPlay is more robust to noisy smart meter data than the FHMM-based approach.

As expected, the error factors increase as we add more circuits and more complexity to a home's data. We also see that the freezer's accuracy is nearly a factor of two higher when including its full set of identifiable features, compared to restricting it to only step features. However, beyond a complexity of 1000 power deltas, the error factors stay roughly constant (with the exception of the refrigerator), even when the complexity goes to 50,000 power deltas. The refrigerator's accuracy decreases significantly when adding a complex load, e.g., in this case a heat recover ventilator that exhibits stable power oscillations. The reason is that its cycle detector is unable to select spikes that correspond to the refrigerator, due to the heat recovery ventilator generating a large number of similarly-sized spikes at various intervals.

Figure 5.12 then examines three specific points from the previous graph and compares them with the FHMM approach. In Figure 5.12(a), we use both PowerPlay and the FHMM approach to track a load from data that *only* includes that load. As shown, the FHMM approach is nearly perfect, since its model is trained on the actual data we disaggregate in this case. By comparison, PowerPlay shows some error due to the fact that our models, while accurate, only include offline features and not attributes based on when and how long the load operates. However, Figure 5.12(b) and (c) shows the error factor for the same loads if we include every circuit both with (b) and without (c) the complex heat recovery ventilator. Prior work on load disaggregation has generally evaluated their algorithms at small scales, e.g., 5-10 individual loads, that are not representative of the multitude of small and complex loads present in a modern home. Our results demonstrate that PowerPlay performs well even as the number and complexity of loads scales up.

The result shows that PowerPlay is significantly more accurate than the FHMM approach for each load, with the exception of the clothes dryer. While PowerPlay is not more accurate than the FHMM approach at small scales, as in (a), with less "noisy" data, it is significantly more accurate as complexity increases. For example, PowerPlay is nearly perfect at detecting the second-to-second power usage of the toaster even within a highly complex trace. Part of the reason for this, as shown in Figure 5.12(a), is that PowerPlay's model of the toaster is highly accurate. In general, the improvement in error factor for each load over the FHMM approach is greater than 2X (and over 100X in the case of the toaster). Both PowerPlay and the FHMM approach perform well on the clothes dryer because it is large compared to the other loads (∼6kW peak power versus ∼kW peak power), such that the added data complexity does not affect detection.

Figure 5.13 shows time-series power data for our representative loads, as well as the inferred time-series using both PowerPlay and the FHMM-based approach. The figures visually show PowerPlay's per-load accuracy compared to FHMM-based disaggregation.



(a) Actual Usage (b) PowerPlay (c) FHMM

Figure 5.13. Online load tracking in PowerPlay generates an inferred time-series of each tracked load's power data (middle column) that is closer to the load's actual usage (left column) than a recent disaggregation algorithm based on Factorial Hidden Markov Models (FHMM) [37, 34] (right column). This figure visually compares actual power usage, PowerPlay's inferred power usage, and the FHMM approach's inferred power usage for representative loads.

Result: *PowerPlay maintains a low per-load tracking error factor as the number of loads, and their complexity, increases in a home. For the loads in our tracking set, the error factor is generally a factor of two less than a state-of-the-art disaggregation algorithm based on FHMMs.*

5.6.3 Case Study: Demand Response Capacity

Lastly, we consider a real application of scalable, online load tracking, where a utility uses it to monitor aggregate demand response capacity across a neighborhood in real time. In this case, we assume the utility is only able to reduce demand by deferring customers' A/Cs, such that the demand response capacity at any point in time is the amount of power consumed by each active A/C. Thus, to estimate demand response capacity over time, the utility must know: i) what percentage of its customers have active A/Cs? and ii) how much power are they consuming? Utilities currently have no way to estimate such demand response capacity over time.

We assume a utility server collects smart meter data from each home, and runs PowerPlay to track the power usage of customer A/Cs. For our case study, we consider a 10-day period of our deployment home's smart meter data, including a central A/C. To simulate many homes across a neighborhood, we generate 100 virtual homes by randomly time-shifting the A/C's power usage within the smart meter data, which results in 100 distinct homes with different time-varying A/C power usage. PowerPlay then uses our model of the A/C (which includes a mix of the cycle, decay, and step features) to track each home's A/C power usage.

We use PowerPlay's output to query the set of active A/Cs across homes over time. For example, at a random point in time, 34 of the 100 homes had an active A/C, with PowerPlay correctly identifying the status of each A/C with 96% accuracy. In particular, PowerPlay detected 30 out of 34 active A/Cs and all inactive A/Cs, demonstrating 88% recall and 100% precision. Of the 30 detected A/Cs, PowerPlay's second-to-second inferred power readings differed from the A/Cs actual power usage by an average of 104W (out of its 3kW average power). PowerPlay estimated the total A/C power usage across the neighborhood, i.e., its demand response capacity, to be 78.1kW, which differs from the actual capacity of 87.9kW by 12%, with the difference primarily due to the four undetected active A/Cs. Excluding the undetected A/Cs, the total A/C power inferred by PowerPlay differed from the actual power by less than 1%.

Result: *PowerPlay enables new applications for online analytics on smart meter data—in this case accurate, online estimation of the grid's demand response capacity.*

5.7 Summary

This chapter presents PowerPlay, a system for online load tracking that emphasizes both efficiency and accuracy. In essence, "tracking" a particular load creates a *virtual power meter* for it, which mimics having a network-connected energy meter attached to it. PowerPlay takes a model-driven approach to online load tracking, which focuses on detecting a small number of identifiable load features in smart meter data. We enumerate an identifiable set of features common across loads, and then design methods to efficiently detect them in smart meter data. By using a high-level feature abstraction, PowerPlay enhances computational tractability, enabling efficient and accurate online load tracking. Our results show that PowerPlay is able to track loads in near real-time, even on low-power embedded platforms, and improves per-load accuracy by a factor of two compared to a FHMM-based disaggregation algorithm.

CHAPTER 6

AUTOMATIC LOAD IDENTIFICATION

The tracking problem detailed in Chapter 5 is primarily of interest when only aggregate whole-house data is available (as in the common case when a building-level smart meter is the sole source of monitoring). When energy data from individual devices is available directly (such as from "smart outlets"), this problem is less important, as devices can be trivially tracked by reading their associated energy data. However, such an environment introduces new challenges, such as maintaining metadata about each monitored outlet (e.g., "outlet #8 is connected to a toaster"). Manually maintaining this information adds significant human overhead and is error-prone when devices are moved; thus, automatic approaches are desirable. In this chapter, we define this problem as performing "non-intrusive load identification" (NILI) and details a technique for automatically identifying devices attached to smart outlets using off-the-shelf classifiers.

6.1 Background and Motivation

Despite the substantial attention given to analysis techniques focusing only on whole-house energy data, decreasing costs of embedded networked sensors has increased the feasibility of outlet-level instrumentation and metering. If these trends continue, we expect that buildings in the near future will be able to install "smart" outlets, which monitor and transmit an outlet's power usage in real time, for nearly the same cost as conventional "dumb" outlets. Examples of smart outlet-level meters that are now widely available include the Belkin WeMo Insight Switch [59], the Insteon iMeter Solo [25], the Budderfly controllable outlet [11], and the Z-Wave Smart Energy Switch [62]. Typically, companies also provide dashboard software to collect and record outlet-level data, and allow users to view it. As a whole, the mainstream home automation sector, which includes both the low-cost smart sensors above, as well as outlets capable of remote load control, is expected to grow by 60% from 2012 to 2018 [10].

While the widespread deployment of low-cost outlet-level meters will provide new visibility into building energy consumption, this shift also raises new challenges related to managing a large and diverse sensor deployment. In particular, since the meters above are built into general-purpose outlets, rather than devices themselves, users must manually identify each specific device plugged into each meter and then update the outlet's meta-data in the dashboard software any time someone plugs a new device into an outlet. For example, if someone plugs a toaster into an outlet, a user must manually associate the outlet with the toaster. The correct association is important, since an particular device might also be associated with other useful attributes, such as its degree of scheduling flexibility or its peak power consumption. While some static outlets may power the same device for long periods of time, as with a refrigerator, many outlets are dynamic and frequently changing due to the use of transient devices, including laptops, vacuums, seasonal air conditioners, and niche kitchen appliances. Even for static outlets attached to the same device, users must still correctly enter the device's name into the dashboard software during setup, which often requires manually recording an obscure outlet identifier—often printed on the back of each outlet—with each device prior to installation.

Such manual identification is both cumbersome and error-prone: users often do not enter any per-outlet meta-data, and whatever meta-data they do enter is either too general to be useful, e.g., "living room outlets," or is never updated and quickly becomes stale. Ultimately, meta-data errors reduce the usefulness of the data to automated management systems and operators. Thus, rather than require users to manually enter device meta-data, we propose a technique for *Non-Intrusive Load Identification* (NILI) that *automatically* identifies new devices plugged into smart outlets without any user intervention.

Unlike the related problem of non-intrusive load monitoring (NILM), NILI has received comparatively little attention. However, NILI is likely to remain relevant into the future, since embedding sensing into general-purpose outlets is more cost-effective than relying on sensors embedded into devices themselves. While the sensor meta-data is constant in the latter case (since each sensor is tightly coupled to the device), this approach requires a sensor for each *device* rather than a sensor for each *outlet*. In addition, outlets are easily standardized during building construction and management, while reliance on third-party manufacturers for device-level support is likely to introduce additional complexities, such as differing sensor hardware capabilities, data formats, and network protocols.

Our approach to NILI is based on the categorization of outlet-level timeseries power data using standard classification techniques. We first perform training on a set of input devices representing basic device types to construct the classifier, using a set of features chosen to accurately distinguish between many devices. The input classes may either be models for specific device types, e.g., a specific type of GE refrigerator, or general models of broad device classes, e.g., a generic refrigerator of any type. During runtime, we periodically ingest recent data from each smart outlet to the classifier and then update the mapping in the outlet meta-data table based on the classifier's output, which specifies the type of device plugged into the outlet. We then evaluate each classifier using a dataset of labeled device energy data collected from several homes, and consider identification of both previously seen and unseen devices. We find that our classifier can achieve classification accuracy of over

90% on our sample dataset, even with a relatively small and straightforward set of classification features.

6.2 Problem Statement and Approach

Formally, we define the NILI problem for a smart outlet O_i as inferring the name of the device d_j plugged into O_i at time t, given the outlet's average power usage $p_i(t)$ each $(t - \tau, t]$. Equivalently, NILI computes the function $O_i(t) \in \{name_j\}$ $\forall j$ and $t > 0$, given $p_i(t)$. NILI assumes a table ("database") of known devices and the key energy characteristics (e.g., distinguishing features) of each device. The table may be either general, including only coarse features that distinguish one type of device from another, or highly specific, including detailed features that distinguish two different models of the same device. In addition to the features, the table may also include other meta-data associated with the device useful for a building management system, such as a device's peak power (which may also be a feature) or its degree of elasticity, i.e., how far in the future a scheduler may shift its power usage without violating its operating constraints, such as maintaining temperature within a specified guardband.

As might be expected, selecting the important features for each device is critical; we discuss feature selection in the next section. A smart outlet's sampling interval τ also affects NILI accuracy. In general, a longer sampling interval "averages out" features in $p_t(t)$, revealing fewer identifiable features and decreasing NILI's accuracy, while a shorter interval reveals more identifiable features and increases accuracy. Our work specifically targets the consumer-grade smart outlets mentioned in the previous section, which commonly provide a sampling resolution on the order of seconds, e.g., $\tau \sim$ seconds.

Figure 6.1 depicts the software and hardware architecture for a building management system that includes a NILI controller, which dynamically updates the meta-data for each of the building's outlets. Specifically, for each of $1 \ldots k$ smart outlets,

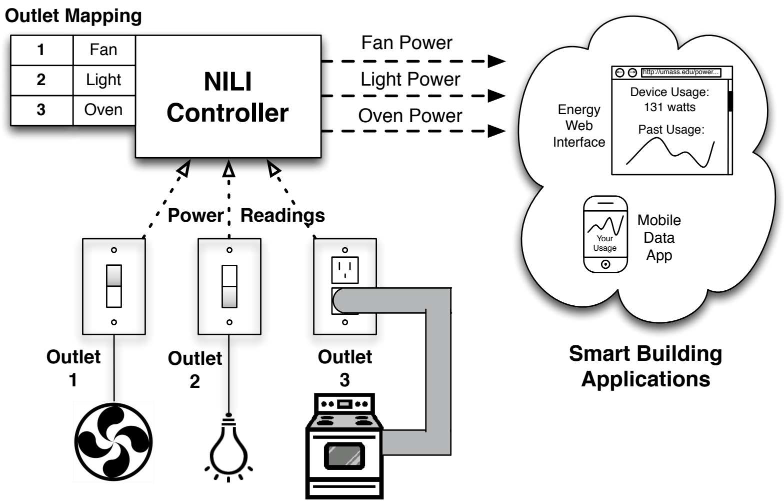


Figure 6.1. Software and hardware architecture for NILI-enabled smart building.

the controller continuously receives energy data transmitted by the outlet and analyzes it to determine the attached device and update the device name associated with the outlet in the meta-data table. The table represents only the current mapping of outlets to devices; our NILI controller also stores all prior mappings by annotating each outlet's timeseries of power data to record each time t a certain device attaches or detaches from an outlet. These annotated data streams can then be used by higher-level data-driven applications, e.g., such as schedulers [8] that use each load's power usage data to determine which loads to defer and when. These data-driven applications require sensors attached to each device, and often implicitly assume a static mapping (or tight coupling) between the sensor and the device. However, as we discuss in Section 1, such a static mapping is usually not feasible in practice.

Our general approach to determining the device attached to each outlet is to train a timeseries classifier on historical power data for each device. The classifier uses the training data to learn an association between the device and the high-level features of its timeseries power usage. Once trained, the classifier simply outputs a device name for a fixed set of consecutive energy readings from each outlet. We represent a set of consecutive energy readings as a series of three-tuples that specify a timestamp, outlet, and average power in watts over τ. Since the device name may change, the controller periodically re-executes the classifier on new outlet data. The interval at which the controller updates each outlet's mapping may be either static, e.g., once every 10 minutes, or dynamic, e.g., based on sudden changes in an outlet's power usage.

6.3 Identification Algorithm

Before classifying an outlet's timeseries of energy readings, our NILI algorithm first converts them into a small set of features that serve as inputs to the classifiers. As discussed below, this process requires first preprocessing the raw timeseries data, then extracting the necessary features for classification, and finally applying various classifiers to the feature set.

Preprocessing. The raw input data consists of average power readings every τ seconds from a smart outlet. We store these power readings, since the classifier operates over a rolling window of historical data. The length of the window necessary to classify an outlet is device-dependent: some device behavior is distinctive enough to classify within seconds of being turned on, e.g., a microwave, while other devices may require multiple duty cycles to discern a distinctive pattern of usage, e.g., a refrigerator. To aid in feature extraction, we preprocess the raw timeseries by computing a timeseries of *energy deltas*, or the difference between two consecutive power readings. Analyzing energy deltas is common in NILM algorithms, since the size of a delta is device dependent, e.g., a 60W power increase due to a light bulb being turned on, and not affected by a building's aggregate absolute energy usage.

Thus, storing and operating on energy deltas is useful for filtering background noise due to the energy usage of a power strip or the smart outlet, itself. Additionally, since consecutive deltas of the same direction, e.g., +40W followed immediately by +20W) often result from changes in power usage occurring across a measurement boundary, we collapse them in a single delta, e.g., +60W. Preprocessing the data to consider such steps as single energy deltas provides a more accurate representation of changes in device power usage, especially given that most energy deltas are zero, i.e., there is no change in power.

Feature extraction. Given a recent window of raw timeseries power data and energy deltas from preprocessing, we next compute a feature vector that captures the behavior of the device. While many features are possible given the input data, we choose a compact set of features that are both intuitive and easily derived directly from the input data.

1. **Statistical Metrics**. The simplest set of features consist of simple statistical metrics of the timeseries power data, including the average power, variance, maximum power, and minimum power over the input time interval. Since infrequently used devices often consume no power, thereby skewing the average power towards zero, we exclude measurements under a threshold value to ensure that we only taking into account periods when the device is operating.

2. **Duty Cycle**. The duty cycle feature is useful for distinguishing continuously operating devices, e.g., an air conditioner, from devices that typically operate only for short periods, e.g., a toaster. We capture a device's duty cycle as the proportion of time is operates over the input interval, called the 'on ratio,' calculated as the number of average power readings over the threshold wattage above divided by the total number of readings.

3. **Waveform**. The most distinguishing feature of a device's power usage is its complete waveform, which represents a long sequence of specific changes in power specific to the device. For example, inductive devices such as the refrigerator and freezer pictured previously in Figure 4.3 includes a large spike followed by a decrease in power to a steady state. We indirectly capture the

waveform as a feature by separating energy deltas into different bins, where the size of each bin represents a distinct feature. We bin the energy deltas as follows: for ten distinct bin sizes ranging from 5W to 500W (with most bin sizes in the < 100W range), we filter the energy deltas to include only changes in average power ranging from the bin size to 5 times the bin size (e.g., 25W to 125W).

For each bin size, we calculate three features, resulting in 30 features total, as follows: (a) the number of changes in average power in the filtered set of energy deltas, (b) the average time interval between steps in the filtered energy deltas, and (c) the number of 'spikes' in the filtered energy deltas, where a 'spike' is defined as a positive step of at least 10 times the bin size, followed immediately by a negative step of at least 30% of the magnitude of the positive step. As discussed previously in Chapter 4, a spike is simply a large but very brief period of energy use caused by the inrush current when a device turns on, and prominently occurs in many kinds of motor-driven devices.

Classification. Finally, we pass the vector of computed device features to a classifier, which returns the inferred device name. The classifier output may either be a general device type, e.g., refrigerator, or a specific device model, e.g., a particular refrigerator manufacturer and model. We evaluate the three different well-known classification algorithms below.

1. **Naive Bayes**. We first consider a simple naïve Bayes algorithm for classification. The key assumption made in naïve Bayes classifiers is the independence of all features, i.e., each feature is conditionally independent of every other feature given the class. Through the application of Bayes' theorem, the conditional distribution over the classes C, i.e., the device types, given the features $f_1, f_2, \ldots, f_n$ is defined by:

$$P(C|f_1, \ldots, f_n) \propto p(C) \prod_{i=1}^{n} P(f_i|C)$$

We use the standard implementation of the naïve Bayes' classifier [30] used in the Weka toolkit [21].

2. **Decision Tree**. We also consider a decision tree classifier [43], which is trained by recursively partitioning the input space, then defining a local model in each resulting region of the input space according to feature values. Although finding the optimal data partitioning is NP-complete, greedy approximation algorithms perform well and benefit from very low training overhead. In our experiments, we use J48, an implementation of the C4.5 decision tree algorithm [50] used in Weka.

3. **Support Vector Machines**. Finally, we consider a classifier using support vector machines (SVMs), a more complex algorithm based around mapping the input feature space into a second, linearly separable feature space using a kernel function. We use the `libSVM` implementation [13] of SVMs supported through Weka. Our reported results in Section 6.4 use a polynomial kernel of degree 2, which was chosen after experimentation with several different kernels.

6.4 Evaluation

We evaluate our algorithm using 1-second data collected from our smart home deployment (described in Chapter 3). We train our classifiers using a dataset gathered from several dozen devices collected over a three month period. For each device over the three month period, we first split the data into 24-hour blocks, then compute a feature vector over each day-long period as described in Section 6.3. Thus, each device results in roughly 90 instances used in training, though we exclude days in which devices went completely unused.

We consider two scenarios – identifying specific devices models, e.g., a specific refrigerator, or identifying general device types, e.g., any refrigerator. The primary advantage of the latter approach is the ability to generalize to previously unseen devices; while the classifier can only output the specific models that it has observed in training, returning device types allows classification of devices not represented in the training dataset. In practice, we envision training a classifier on a very large dataset of devices collected from many homes, then using that classifier on both existing and new devices not present at the time of training. Although our evaluation dataset is relatively small, i.e., 3 instances of most major appliance types, we consider both approaches to illustrate the potential of unseen device identification.

Identification of previously seen devices. We first consider classification performance when training on the complete dataset – i.e., identification of previously seen devices. For each of the three classifiers—Naive Bayes, C4.5, and `libSVM`—we perform 10-fold cross-validation on the dataset to quantify identification accuracy both for specific devices and for device types. The results are shown in Figure 6.2. We see that accuracy is quite high in all scenarios – both C4.5 and SVM demonstrate accuracy of over 90% for device types, with naïve Bayes somewhat lower at roughly 70%. Performance on specific device identification is modestly lower than for general device types. The difference stems from the ability of the classifier to generalize the properties of the device types, e.g., a refrigerator, given a broader training dataset, as well as the smaller number of possible classes. However, this different only amounts to less than 10% in all cases.

Breakdown by device and type. The results in Figure 6.2 demonstrate the overall performance of the classifiers, but classification accuracy may vary significantly from device to device, due to the presence of unique characteristics, or lack thereof, reflected in the feature vector. For example, refrigerators have a regular cyclic power usage pattern, which typically results in a high ‘on ratio’, while most electronic loads

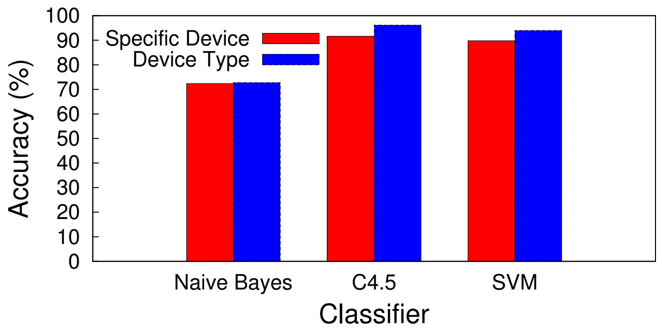


Figure 6.2. Classification performance on the entire dataset, i.e., identifying previously seen devices, using 10-fold cross-validation. Both performance on specific device instances and general device classes is shown.

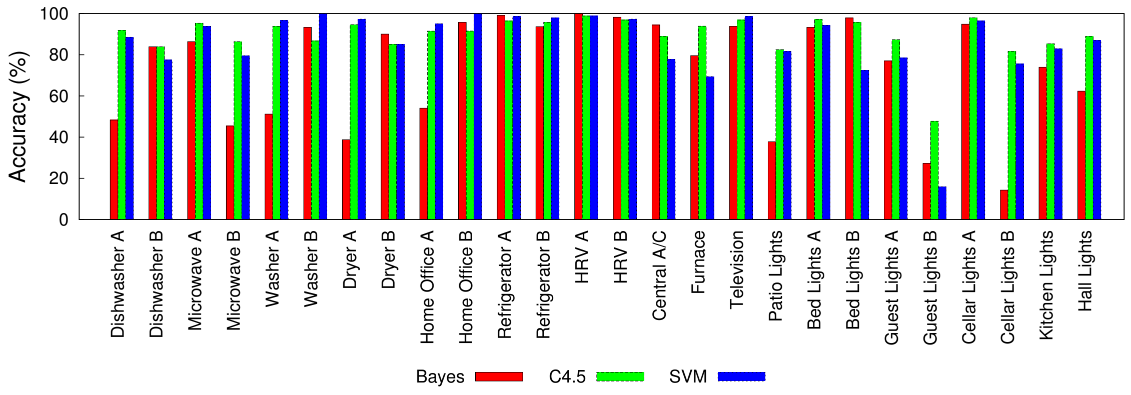


Figure 6.3. Individual device identification accuracy per device. Devices A and B represent two instances of the same device type (with different specific models).

have highly erratic power consumption, due to the variable behavior of switch-mode power supplies, which typically results in higher power variance than other types of devices.

Figure 6.3 shows the individual, device-level classification accuracy for a subset of our devices. As in Figure 6.2, the best performance for nearly all devices is observed with the C4.5 or SVM classifiers, with accuracy of over 95% for many devices. The performance of the naïve Bayes classifier, on the other hand, is inconsistent, with some devices showing quite poor performance (many less than 50%) – in these cases, the naïve Bayes classifier has difficulty distinguishing between multiple instances of

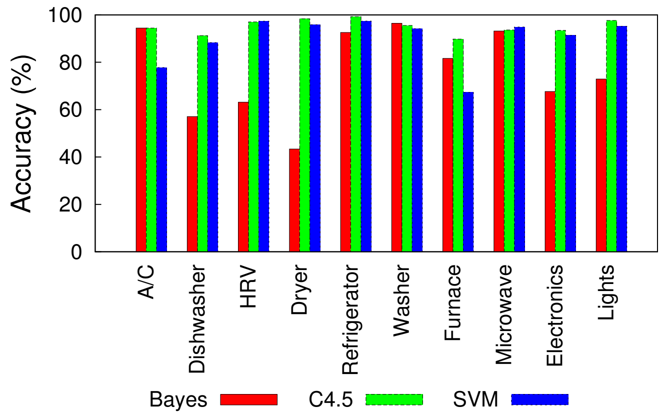


Figure 6.4. General device type identification accuracy, broken down by type.

the same device type, e.g., multiple dishwashers or multiple dryers. As a result, accuracy on one such instance of a given type remains high, while performance on other instances of that type is low (as these instances are identified as the 'dominant' first instance).

Figure 6.4 shows the corresponding results broken down by device types, rather than individual devices. Performance is more consistent in this case, although naïve Bayes continues to show significantly lower performance for certain device classes, such as dishwashers and clothes dryers. These types of devices exhibit more complex behavior than most of the other device types, e.g., as indicated by the washing machine's average power trace in Figure 4.10, which implies that the simplistic naïve Bayes classifier is not able to identify them as accurately as the more sophisticated C4.5 and SVM classifiers.

Identification of previously unseen devices. Finally, we consider the case where we wish to identify devices that have *not* been previously observed during training. This approach limits us to identifying device types rather than specific device models, as it is impossible to generate a class label that was not seen during training. For this experiment, we trained our classifier on devices within two of the three houses, then attempted to classify devices in the third house, which are not

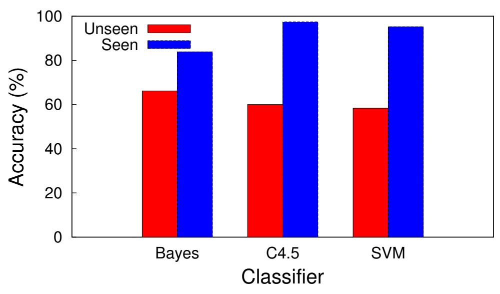


Figure 6.5. Accuracy of device type identification on previously ‘unseen’ devices, along with accuracy when all devices are in the training dataset (‘seen’).

represented in the training data. As before, we report the 10-fold cross-validation accuracy of identification.

Overall identification accuracy is shown in Figure 6.5. For comparison, these results are shown alongside results when all devices are included in training (reproduced from Figure 6.2). Unsurprisingly, identification accuracy falls substantially, as we are relying strictly on the ability of the classifier to distill the essential properties of the device *type* rather than any specific device instance. Accuracy of both the C4.5 and SVM classifier fall to below 60%. Interestingly, the naïve Bayes classifier degrades substantially less (to 65%) and actually outperforms the other classifiers, reversing the trend seen when identifying previously seen devices. This result suggests that the simple naïve Bayes classifier more effectively generalizes the device type, but at the expense of distinguishing specific instances of device types (as seen previously in Figure 6.3). Furthermore, while the absolute result of 65% is not particularly high, we stress that we are attempting to generalize the device type given a very limited set of training instances (just two in most cases), so we view these results as encouraging and would improve with more training data.

Finally, Table 6.1 shows the confusion matrix for the classification of unseen devices using the C4.5 classifier (i.e., the third bar of Figure 6.5). We see that there is a wide variation in the accuracy of identification of the various device types. For

	Dryer	**Fridge**	**Washer**	**MWave**	**Electronics**	**Light**
Dryer	9	0	0	0	0	0
Fridge	1	99	0	34	21	45
Washer	44	0	5	5	0	0
MWave	3	1	0	53	0	1
Electronics	0	0	0	0	2	14
Light	0	0	0	0	0	86

Table 6.1. Confusion matrix for the decision tree classifier on unseen devices (rows are actual, columns are predicted).

example, every light is correctly identified as a light (i.e., perfect recall), which is understandable given the flat energy profile of nearly all lights. The same is true of the microwaves, which as short-lived but high-power devices are easily identified. The washing machines, on the other hand, are frequently misidentified as dryers – both types are large, sporadically active devices with complex and highly variable power signatures, and as such it is difficult for the classifier to distinguish the two. A significant portion of the overall classifier error comes from the poor performance of the Electronics device type (nearly all of which are identified as lights), likely due to the fact that there are many different types of electronics displaying differing power signatures. Regardless, we see that the classifier is generally able to accurately distill device types with unique energy characteristics and use those characteristics to identify unseen devices.

6.5 Summary

In this chapter, we considered the problem of Non-Intrusive Load Identification (NILI), in which devices connected to outlet-level energy meters, i.e., smart outlets, are automatically identified, alleviating the user from the cumbersome and error-prone task of manually maintaining meta-data on specific devices and outlets. We propose an approach to performing NILI that transforms energy time-series data into a compact set of intuitive features, then uses an off-the-shelf classifier to identify unknown devices. Using a dataset of device energy traces collected from three homes, our experiments demonstrate that we can achieve greater than 90% accuracy on devices represented in training data. Furthermore, even with a small sample of devices of a given type, e.g., refrigerators, we are often able to identify previously *unseen* devices as particular types of devices, demonstrating the ability of the classifier to generalize the properties of device types. As future work, we plan to consider other features (for example, featured computed based directly on the models detailed in Chapter 4) and a larger set of training devices to further evaluate NILI's potential.

CHAPTER 7
SUMMARY

7.1 Thesis Summary

This thesis has discussed techniques for analyzing energy meter data to provide insights in next-generation smart homes. In particular, we have made the following contributions:

Smart home architecture. First, we proposed a simple but flexible architecture for smart homes, combining low-cost monitoring and analysis. We overviewed our own real-world smart home deployment following this approach, which provides the foundation for our analytics work. We also discussed the challenges of combining monitoring and control capabilities in smart homes using highly-available home automation (HA) protocols.

Load modeling. Second, we presented a flexible modeling framework derived from the core electrical properties of devices. Our modeling framework is able to efficiently describe nearly all household loads using a compact set of basic models. We derived our models from an empirical study of smart meter data and demonstrated that our models are more accurate than simplistic 'on-off' device models used in prior work.

Online load tracking. Third, we proposed technique for performing online load tracking, which differs from traditional disaggregation in its focus on online operation, single-load accuracy, and system scalability. Our technique tracks devices using parameterized models derived from our modeling framework, and uses a set of feature detectors to extract recognizable features (and the devices they represent) from within a noisy, aggregate smart meter trace. In doing so, we are able to present the abstraction of a 'virtual power meter', which is useful for many types of applications. We evaluated our system and found that it (a) is more accurate at fine timescales than traditional disaggregation techniques, and (b) is capable of scaling to utility scales (e.g., hundreds of homes) while maintaining nearly real-time operation.

Non-intrusive load identification. Fourth, we defined the problem of *non-intrusive load identification* as automatically assigning meta-data (e.g., device names) to smart outlets. We proposed a technique using off-the-shelf classifiers to automatically identify devices attached to such outlets and demonstrated that we can effectively maintain a dynamic mapping of devices to outlets, which is important in providing feedback to users and understanding how homes actually using energy.

7.2 Future Work

The work covered in this thesis naturally points towards several areas of future work, several of which are outlined below.

Supplemental data sources. While this thesis has focused almost exclusively on real power data for performing analytics, real-world environments (including our own deployment) provide many different types of data that may be useful. For example, most of the techniques described previously may potentially be enhanced by considering both real and reactive power. Additionally, many types of environmental data are readily available, such as occupancy information, door events (e.g., when a door is opened or closed), and weather data (such as temperature and humidity). These sources of information are often invaluable in predicting and explaining device behavior (e.g., the interior temperature of a room will correlate very closely with the operation of an air conditioner).

Models of user behavior. The device models described in Chapter 4 are explicitly designed to be user-agnostic – i.e., the model is specific to the device, but *not* to the user of the device. This approach has the benefit of allowing a model to generalize across all usage patterns, but also means that these usage patterns are not exploited even if they are highly predictive. As real-world device usage is a function of both the device itself and of user patterns, considering both inputs explicitly (i.e., a model of the device *and* a model of user behavior) may lead to greater accuracy in smart home analytics.

Applications of analytics. While analytics such as device tracking and identification are important components of automated smart homes, we ultimately wish to realize useful end-user applications (such as lowering a homeowner's electricity bill via device rescheduling). The analytic techniques discussed in this thesis are largely a prerequisite to these types of applications, and open the door towards many interesting problems in smart buildings, such as device scheduling, identifying and incentivizing energy savings, and preserving user privacy in the face of sophisticated data collection and analysis.

In summary, this thesis has explored several important problems in enabling efficient, automated data analysis in smart homes. Accurate and reliable analytics are a key milestone towards realizing real-world smart homes, and the techniques proposed here take several steps in this direction.

BIBLIOGRAPHY

[1] Aeon Labs Z-Wave Smart Energy Switch. `http://aeotec.com/z-wave-plug-in-switch`.

[2] Anderson, K., Ocneanu, A., Benitez, D., Carlson, D., Rowe, A., and Berges, M. BLUED: A Fully Labeled Public Dataset for Event-Based Non-Intrusive Load Monitoring Research. In *SustKDD* (August 2012).

[3] Ardakanian, Omid, Keshav, S., and Rosenberg, C. Markovian Models for Home Electricity Consumption. In *GreenNets* (August 2011).

[4] Armel, K., Gupta, A., Shrimali, G., and Albert, A. Is Disaggregation the Holy Grail of Energy Efficiency? the Case of Electricity. *Energy Policy 52*, 1 (January 2013).

[5] Banerjee, N., Rollins, S., and Moran, K. Automating Energy Management in Green Homes. In *HomeNets* (August 2011).

[6] Barker, S., Kalra, S., Irwin, D., and Shenoy, P. Empirical Characterization and Modeling of Electrical Loads in Smart Homes. In *IGCC* (June 2013).

[7] Barker, S., Mishra, A., Irwin, D., Cecchet, E., Shenoy, P., and Albrecht, J. Smart*: An Open Data Set and Tools for Enabling Research in Sustainable Homes. In *SustKDD* (August 2012).

[8] Barker, S., Mishra, A., Irwin, D., Shenoy, P., and Albrecht, J. SmartCap: Flattening Peak Electricity Demand in Smart Homes. In *PerCom* (March 2012).

[9] Berardino, Jonathan, and Nwankpa, C. Dynamic load modeling of an hvac chiller for demand response applications. In *SmartGridComm* (October 2010).

[10] Brennan, M. House of the future: How automation tech is transforming the home. In *Forbes* (October 2013).

[11] `http://www.budderfly.com/`, May 2014.

[12] Carpenter, T., Singla, S., Azimzadeh, P., and Keshav, S. The Impact of Electricity Pricing Schemes on Storage Adoption in Ontario. In *e-Energy* (May 2012).

[13] Chang, Chih-Chung, and Lin, Chih-Jen. Libsvm: a library for support vector machines. *TIST 2*, 3 (2011), 27.

[14] U.S. Department of Energy. Building Energy Data Book. http://buildingsdatabook.eere.energy.gov/, 2010.

[15] eGauge Energy Monitoring Solutions. `http://www.egauge.net/`, 2013.

[16] U.S. Energy Information Administration, Frequently Asked Questions, How Many Smart Meters are Installed in the U.S. and who has them? `http://www.eia.gov/tools/faqs/faq.cfm?id=108&t=3`.

[17] Forney, G. The Viterbi Algorithm. *Proceedings of the IEEE 61*, 3 (1973), 268–278.

[18] Froehlich, J., Larson, E., Gupta, S., Cohn, G., Reynolds, M., and Patel, S. Disaggregated End-Use Energy Sensing for the Smart Grid. *IEEE Pervasive Computing, Special Issue on Smart Energy Systems 10*, 1 (January 2011).

[19] Gupta, S., Reynolds, M., and Patel, S. ElectriSense: Single-Point Sensing Using EMI for Electrical Event Detection and Classification in the Home. In *Ubicomp* (September 2010).

[20] Guruplug server. `https://www.globalscaletechnologies.com/t-guruplugdetails.aspx`.

[21] Hall, Mark, Frank, Eibe, Holmes, Geoffrey, Pfahringer, Bernhard, Reutemann, Peter, and Witten, Ian H. The weka data mining software: An update. *SIGKDD Explor. Newsl.* (Nov 2009).

[22] Hart, G. Residential Energy Monitoring and Computerized Surveillance via Utility Power Flows. *IEEE Technology and Society Magazine 8*, 2 (June 1989).

[23] Hart, G. Nonintrusive Appliance Load Monitoring. *IEEE 80*, 12 (December 1992).

[24] Hnat, T., Srinivasan, V., Lu, J., Sookoor, T., Dawson, R., Stankovic, J., and Whitehouse, K. The Hitchhiker's Guide to Successful Residential Sensing Deployments. In *SenSys* (November 2011).

[25] imeter solo. `http://www.insteon.net/2423A1-iMeter-Solo.html`.

[26] Insteon: The Details. `www.insteon.net/pdf/insteondetails.pdf`, 2005.

[27] Insteon for the Smart Grid. `http://www.insteonsmartgrid.com`.

[28] Jiang, X., Dawson-Haggerty, S., Dutta, P., and Culler, D. Design and Implementation of a High-Fidelity AC Metering Network. In *IPSN* (2009).

[29] Jiang, X., Ly, M. Van, Taneja, J., Dutta, P., and Culler, D. Experiences with a High-Fidelity Wireless Building Energy Auditing Network. In *SenSys* (2009).

[30] John, George H., and Langley, Pat. Estimating continuous distributions in bayesian classifiers. In *Eleventh Conference on Uncertainty in Artificial Intelligence* (San Mateo, 1995), Morgan Kaufmann, pp. 338–345.

[31] Kelly, D. Disaggregating Smart Meter Readings using Device Signatures. In *Masters Thesis, Imperial College London* (2011).

[32] Kelso, J., Ed. *2011 Buildings Energy Data Book*. Department of Energy, March 2012.

[33] Keogh, E., and Pazzani, M. Dynamic Time Warping with Higher Order Features. In *SDM* (April 2001).

[34] Kim, H., Marwah, M., Arlitt, M., Lyon, G., and Han, J. Unsupervised Disaggregation of Low Frequency Power Measurements. In *SDM* (April 2011).

[35] Kim, Y., Schmid, T., Charbiwala, Z., and Srivastava, M. ViridiScope: Design and Implementation of a Fine Grained Power Monitoring System for Homes. In *Proceedings of the Conference on Ubiquitous Computing (UbiComp)* (Orlando, Florida, September 2009), pp. 245–254.

[36] Klingensmith, N., Willis, D., and Banerjee, S. A Distributed Energy Monitoring and Analytics Platform and its Use Cases. In *BuildSys* (November 2013).

[37] Kolter, J., and Johnson, M. REDD: A Public Data Set for Energy Disaggregation Research. In *SustKDD* (August 2011).

[38] Kolter, J., and Ng, A. Energy Disaggregation via Discriminative Sparse Coding. In *NIPS* (December 2010).

[39] Koomey, J. Data Center Electricity Use 2005 to 2010. Tech. rep., Analytics Press, August 2011.

[40] Lanzisera, Steven. The "Other" Energy in Buildings: Wireless Power Metering of Plug-in Devices. Environment Energy Technologies Division Seminar, Lawrence Berkeley National Labs, June 17 2011.

[41] Levenberg, K. A Method for the Solution of Certain Non-Linear Problems in Least Squares. *Quarterly of Applied Mathematics* (1944).

[42] Marchiori, A., and Han, Q. Using Circuit-Level Power Measurements in Household Energy Management Systems. In *BuildSys* (2009).

[43] Murphy, Kevin P. *Machine learning: a probabilistic perspective*. The MIT Press, 2012.

[44] Pannuto, P., and Dutta, P. Exploring Powerline Networking for the Smart Building. In *IP+SN* (2011).

[45] Patel, S., Robertson, T., Kientz, J., Reynolds, M., and Abowd, G. At the Flick of a Switch: Detecting and Classifying Unique Electrical Events on the Residential Power Line. In *Ubicomp* (September 2007).

[46] Pacific Gas and Electric, leveraging Smart Meter Installation Progress. `http://www.pge.com/myhome/customerservice/smartmeter/deployment/`, February 2013.

[47] Plaisance, Mike. Holyoke gas and electric customers could net savings from ongoing meter study involving high performance computing center. `http://www.masslive.com/news/index.ssf/2014/04/holyoke_gas_and_electric_custo.html`.

[48] Plogg Wireless Energy Management. `http://www.plogginternational.com`.

[49] The Power Consumption Database. `http://www.tpcdb.com`.

[50] Quinlan, John Ross. *C4. 5: programs for machine learning*, vol. 1. Morgan kaufmann, 1993.

[51] Reinhardt, A., Baumann, P., Burgstahler, D., Hollick, M., Chonov, H., Werner, M., and Steinmetz, R. On the accuracy of appliance identification based on distributed load metering data. In *SustainIT* (2012).

[52] Ridi, A., Gisler, C., and Hennebert, J. Automatic identification of electrical appliances using smart plugs. In *WoSSPA* (May 2013).

[53] Ruzzelli, A.G., Nicolas, C., Schoofs, A., and O'Hare, G. M P. Real-time recognition and profiling of appliances through a single electricity sensor. In *SECON* (2010).

[54] Taneja, J., Culler, D., and Dutta, P. Towards Cooperative Grids: Sensor/Actuator Networks for Renewables Integration. In *SmartGridComm* (2010).

[55] Energy, Inc. `http://www.theenergydetective.com/`, July 2012.

[56] Tweet-a-Watt. http://www.ladyada.net/make/tweetawatt/.

[57] Smart Meter Implementation Programme. `https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/42737/1480-design-requirement-annex.pdf`.

[58] Viterbi, A. Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm. *IEEE Transactions on Information Theory 13*, 2 (1967), 260–269.

[59] Wemo insight switch. `http://www.belkin.com/us/F7C029-Belkin/p/P-F7C029/`.

[60] Zeifman, M., and Roth, K. Nonintrusive Appliance Load Monitoring: Review and Outlook. *IEEE Transactions on Consumer Electronics 57*, 1 (February 2011).

[61] Zoha, A., Gluhak, A., Imran, M., and Rajasegarar, S. Non-intrusive load monitoring approaches for disaggregated energy sensing: A survey. *Sensors – Open Access Journal 12*, 12 (November 2012).

[62] `http://www.aeon-labs.com/site/products/view/5/`, May 2014.

[63] Zufferey, Damien, Gisler, Christophe, Khaled, Omar Abou, and Hennebert, Jean. Machine learning approaches for electric appliance classification. In *ISSPA* (July 2012).