PowerPlay: Creating Virtual Power Meters through Online Load Tracking

Sean Barker[*]
Bowdoin College
sbarker@bowdoin.edu

Sandeep Kalra, David Irwin, and Prashant Shenoy
University of Massachusetts Amherst
[skalra,shenoy]@cs.umass.edu, irwin@ecs.umass.edu

Abstract

Online load tracking is the problem of monitoring an individual electrical load's energy usage by analyzing a building's smart meter data. The problem is important, since many energy optimizations require fine-grained, per-load energy data in real time; it also differs from the well-studied problem of load disaggregation in that it emphasizes efficient, online operation and per-load accuracy, rather than accurate disaggregation of every building load via offline analysis. In essence, tracking a particular load creates a *virtual power meter* for it, which mimics having a networked-connected power meter attached to it. To enable high performance, we take a model-driven approach that focuses on efficiently detecting a small number of identifiable load features in smart meter data. Our results demonstrate that our system, called PowerPlay, i) enables efficient online tracking on low-power embedded platforms, ii) scales to thousands of loads (across many buildings) on server platforms, and iii) improves per-load accuracy by more than a factor of two compared to a state-of-the-art load disaggregation algorithm.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; J.7 [**Computer Applications**]: Computers in Other Systems—*Command and control*

General Terms

Design, Measurement, Performance

Keywords

Load Monitoring, Load Modeling, Smart Grid

[*] This work was done while the author was at the University of Massachusetts Amherst.

BuildSys'14, November 5–6, 2014, Memphis, TN, USA.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-3143-2/14/11 ...$15.00
http://dx.doi.org/10.1145/2674061.2674068

1 Introduction

Collectively, buildings consume significantly more energy (41%) than society's other broad sectors of consumption—industry (30%) and transportation (29%) [14]. As a result, the design of "smart" buildings that are capable of automatically regulating their energy usage has become an important research area. However, one continuing impediment to improving building energy-efficiency is that, despite much prior research [12], accurate, fine-grained, online monitoring of electrical loads[1] at large scales remains problematic: deploying and maintaining large numbers of embedded networked sensors in every building is prohibitively expensive, invasive, and unreliable. Unfortunately, timely and accurate knowledge of per-load energy usage is a prerequisite for implementing many energy optimization techniques [5, 7, 22].

Rather than rely on expensive instrumentation—via embedded sensors—to monitor loads, an alternative approach is to analyze electricity data from smart meters to infer a load's energy usage. This approach is becoming increasingly attractive, since smart meters, which monitor an entire building's energy usage at small intervals, e.g., minutes to seconds, are now being widely deployed by electrical utilities and consumers [10]. In this paper, 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 creates a *virtual power meter* for it, which 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 [5]. 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.,

[1]We use the term electrical load, or simply *load*, to refer to any distinct appliance or device that consumes electricity.

such as turning off an idle coffee pot [2]. 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.

Our work builds on prior work, which has already developed a variety of analysis techniques for smart meter data, including load disaggregation [1, 11, 18, 26] and occupancy detection [8]. Many startup companies are now combining such energy-based analytics with cloud-based, "big data" platforms [6] to mine building smart meter data *en masse*. However, we argue that online load tracking differs from the well-studied problem of *complete load disaggregation*, often termed Non-Intrusive Load Monitoring (NILM) [1, 11, 18, 26], in two important respects:

Simplicity. Online load tracking is a simpler problem than complete load disaggregation—load tracking targets *individual* loads, while complete load disaggregation focuses on disaggregating an entire building by apportioning its total energy usage across *every* load. Clearly, if complete, accurate, and inexpensive disaggregation was feasible, it would subsume the problem of online load tracking. However, techniques for complete disaggregation continue to suffer from inaccuracy, especially when disaggregating small loads or scaling up to large numbers of loads [1]. Thus, load tracking is better suited for scenarios where disaggregating *all* of a building's loads is either infeasible (due to the large number of loads) or simply not necessary.

Efficiency. Prior disaggregation techniques implicitly assume offline analysis and are often computationally expensive. In contrast, load tracking explicitly targets online monitoring in near real time. This leads us to focus on performance issues not addressed in prior research, such as enabling tracking to either i) run on the low-power embedded platforms used in smart meters or ii) scale to thousands of loads on server platforms.

To enable high performance, we take a model-driven approach to load tracking, which focuses on detecting a small number of identifiable load features in smart meter data. These features derive from a parameterized model of a load's energy usage profile over time, which is based on a small number of fundamental electrical characteristics, i.e., whether a load is resistive, inductive, non-linear, or cyclical. A detailed description of these load types, and their corresponding models, is described in prior work [3]. We select a compact set of identifiable load features from the models and then design efficient online methods for tracking loads by detecting one or more of these features in smart meter data. In doing so, we make the following contributions:

Feature Selection. We describe a compact set of features that loads may exhibit, including power steps, spikes, growths, decays, oscillations, and cycles. We extract a load's features from its model and then choose a small set of identifiable features for tracking. Using only identifiable features to track loads increases efficiency, compared to using every feature, while maintaining accuracy.

Online Load Tracking. For each feature, we design efficient online methods to detect that feature in smart meter data. Since a load may exhibit multiple features, tracking a load may require using multiple feature detectors. Hence, we present an online tracking algorithm that combines multiple feature detectors to efficiently detect and track loads.

Implementation and Evaluation. We implement our load tracking system, called PowerPlay, and evaluate it "live" using a 1Hz power meter. We show that our approach enables efficient, online load tracking: on a 2.4 GHz, single-core server, PowerPlay is able to track loads in smart meter data comprised of nearly 100 loads in real time each second—the same resolution of the building's power meter. We also show that PowerPlay improves per-load accuracy by more than a factor of two compared to a state-of-the-art disaggregation algorithm (based on Factorial Hidden Markov Models (FHMMs) [16, 18]) designed for offline analysis.

2 Background and Approach

PowerPlay assumes a building equipped with a networked power meter that monitors its aggregate electricity usage over time. We refer to this building power meter as a *smart meter*. We assume smart homes employ automated energy management techniques, which require real-time operational knowledge of particular loads' energy usage, e.g., air conditioners (A/Cs), furnaces, or other appliances amenable to automated energy management. Rather than directly monitoring such loads using sensors, our goal is to provide a virtual power meter abstraction that tracks a load's energy usage and when it turns on and off from the home's smart meter data. Load tracking is useful in scheduling home loads or pushing alerts to users (e.g., to indicate that a laundry cycle is complete), or when exercising control over "large" loads (such as A/Cs) across many homes to smooth grid demand.

2.1 Problem Statement

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 $t+\varepsilon$ 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:

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

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 [18].

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. Our work specifically targets consumer-grade power meters, such as the TED [23], eGauge [9], and BrulTech, which commonly provide a sampling resolution of one reading per second, e.g., τ=1 second. While today's utility-grade smart meters provide, at most, minute-level sampling, e.g., a reading once every five to fifteen minutes is common, there are indications the next generation of meters will provide second-level sampling. For example, a U.K. subcommittee defining future smart meter specifications recently released a report advocating a five second sampling resolution [25].

2.2 Prior Work

Our focus on tracking individual loads, rather than complete disaggregation, stems from a recognition that i) accurate disaggregation continues to be an elusive goal despite two decades of research, and ii) the simpler load tracking problem is sufficient for many sensor-based applications and can be more efficient and accurate. Prior disaggregation approaches differ widely based on τ's value, which ranges from $>$100,000,000 samples per second [21] to one sample per hour [19]. Interestingly, a recent survey [1] points out that, despite τ's importance, prior work often does not report it.

In addition, despite the plethora of prior work on disaggregation, the same survey [1] highlights the lack of research that targets second-level sampling. To the best of our knowledge, only Hart's original work [11] and two recent papers [16, 18], which both use an approach based on Factorial Hidden Markov Models (FHMMs), target data with second-level sampling resolution, albeit for full disaggregation. Since there is no prior work on online load tracking, we use a FHMM technique modified for online operation as a baseline "strawman" for comparing PowerPlay's performance and accuracy, as described in §6.

2.3 Basic Approach

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 1 depicts the basic workflow of each subproblem, which we, in turn, outline briefly below.

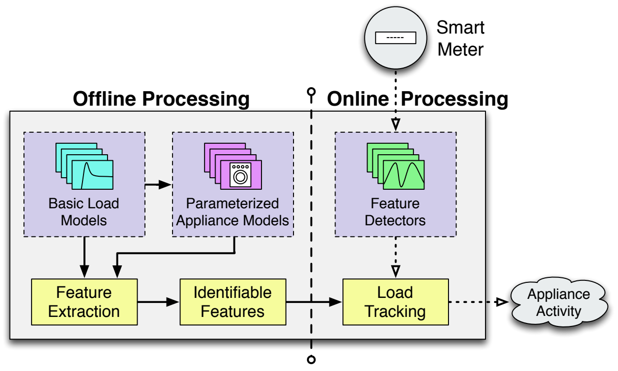


Figure 1. PowerPlay uses offline modeling and feature extraction for online load tracking.

1. Empirical Modeling. We first empirically model each load's energy usage based on properties of the four basic types of electrical loads, i.e., resistive, inductive, capacitive, and non-linear. Prior work describes how to derive such models and shows that such empirical models accurately capture the behavior of nearly every common household load [3]. 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 [13]. 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 (sliding) 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 [15], 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.

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 this process is a one-time step, 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, such as in 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 [24].

3.1 Modeling and Feature Extraction

Electrical loads in an alternating current (AC) system fall into one of four basic types—resistive, inductive, capacitive, or non-linear. Informally, resistive loads include heating elements, such as a toaster; inductive loads include AC motors, such as fans or compressors; and non-linear loads include any type of electronic device, such as TVs or computers. Loads behaves differently based on their load type, but devices of the same type exhibit many common 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. Further details of how the four basic types map to real-world devices are provided in [3]. Below, we enumerate the identifiable features 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 only these simple steps when on.

Power Growth, Decay, and Spikes. Many loads experience smooth increases or decreases in power when turned on (e.g., due to decreasing resistance as a heating element warms), or abrupt and sudden spikes in power (e.g., when starting an induction motor). 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. Many non-linear devices based on electronic controllers (e.g., microwaves) draw a seemingly random amount of power within a fixed range when on. We consider bounded power oscillations between maximum and minimum power thresholds as a distinct feature resembling a random walk between thresholds.

Stable Power Oscillations. Some non-linear loads only have either an upper threshold or a lower threshold, resulting in oscillations from a stable power state (e.g., due to the variable draw of a switched mode power supply). 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. 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.

Since essentially every electrical load is either an induction motor, heating element, non-linear electronics, or some combination thereof, 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 §4. Note that the features above are parameterized for each specific load (e.g., the magnitude of a step or the rate of a decay), and may differ across two loads of the same type, e.g., two A/Cs from different manufacturers may require different features and parameters. Thus, PowerPlay's offline component not only extracts the features of a load, but also determines the parameters for each feature. Figure 2 includes annotated features in power usage data for a variety of common loads.

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 a washing machine, may exhibit an excessively large number of features. Fortunately, searching for every feature is generally not necessary for accurate detection; it is often sufficient to select a subset of prominent features to uniquely identify the load. PowerPlay leverages this insight to only search for a small set of *identifiable features* to match complex loads, which improves both efficiency and scale.

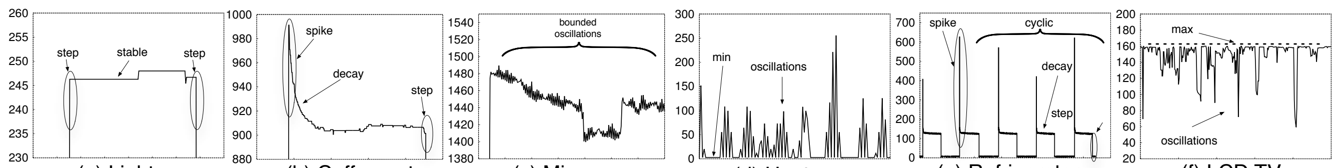
Selecting identifiable features for a load is a one-time offline task, and presents a tradeoff between accuracy and performance. A smaller set of identifiable features improves the efficiency of detection, but decreases tracking's accuracy. At present, we construct a complex load's set of identifiable 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.

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 §3. The right side of Figure 1 depicts this process.

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 (i.e., a sliding window ending with the most recent reading). 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.



(a) Light (b) Coffee pot (c) Microwave (d) Heater (e) Refrigerator (f) LCD TV

Figure 2. Annotated features from representative loads.

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 §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 §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.

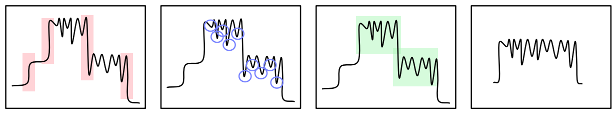

(a) Filter (b) Label (c) Cluster (d) Reconstruct

Figure 3. Detection of a stable oscillation feature.

4.2 Feature Detection

PowerPlay's tracking algorithm relies on individual feature detectors to identify the features described in §3, including power steps, spikes, growth/decay, bounded oscillations, and stable min-max oscillations. We detail each of these feature detectors below.

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 (i.e., a power drop) there is a corresponding positive power delta 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. Power changes that update p are considered background activity, which are excluded from the stable power oscillation feature, while any other oscillations within the window are flagged for consideration in the feature. Finally, we cluster nearby groups of labeled points to result in the time range (and flagged deltas) comprising the feature.

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 illustrated in Figure 3. 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 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

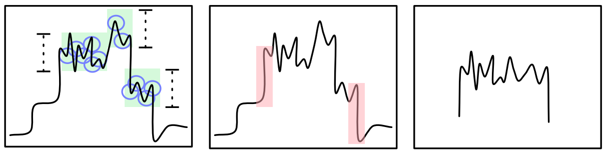


Figure 4. Example of bounded oscillation detector.

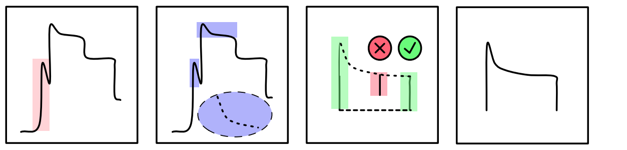


Figure 5. Operation of the decay/growth detector.

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. 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 an initial 15s window, 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 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 4.

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 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) [20] to perform curve fitting. 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, 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 illustrated in Figure 5.

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 standard 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, the trivial approach to identifying them (detecting second-to-second deltas of a certain magnitude) is often inaccurate due to the fact that loads turn on at different points within the sampling interval. Thus, similar to the spike detector above, we collapse multi-second power deltas in the same direction into a single aggregate delta before comparing the step's magnitude against a specific (i.e., parameterized) step feature. Deltas previously assigned to other features are excluded from consideration in this process.

Cycle Detector. 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 6 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. 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 6.

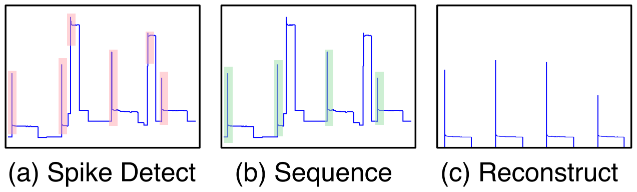


Figure 6. Operation of the cycle detector.

As an 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.

5 Implementation

We implement PowerPlay's feature detectors and tracking algorithm as a library in Perl. The input to the 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 a real home. We describe the home, its loads, and our instrumentation in prior work [4]. Briefly, the home includes a Internet-enabled power meter installed in its electrical panel to monitor the second-to-second power usage of the home and each of its circuits. There are multitude of such meters now available, both commercially [23] and in recent research [17], that record home-level and circuit-level data at 1Hz sampling resolution. We also record ground truth power data (or on-off events, which we correlate with the power meter data) for individual loads not connected to dedicated circuits using either Z-Wave Smart Energy Switches, Insteon iMeters, or Insteon SwitchLincs. In total, our deployment includes 92 sensors producing roughly four million data points per day. Such an extensive deployment is necessary to compare our results, based the home's power data, with ground truth power data from each individual load.

Of course, since our offline modeling and feature extraction methodology is new to this paper, we must manually model each load we track and extract its important features ourselves. 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. We also plan to release the models and extracted features for the loads that we track in §6.

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 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 [18] to evaluate their Reference Energy Disaggregation Dataset (REDD), which is similar to the technique by Kim et al. [16].

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 in real time using per-second power data for the entire home, which operates 92 distinct loads.

6.1 Tracking Efficiency

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.

The tracking efficiency of PowerPlay is determined by the computation overhead of the feature detectors in processing the most recent window of data. This overhead determines both (a) the tracking delay (ε from §2) of the system, where ε=1 second is perfect (1 Hz) real-time tracking, 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. We vary a common window size across all features, then observe the tracking delay (ε) achieved by PowerPlay. As seen in Figure 7, 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, loads with more features (e.g, the dryer) result in a longer 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, most features require significantly less than a 24-hour window to reliably detect.

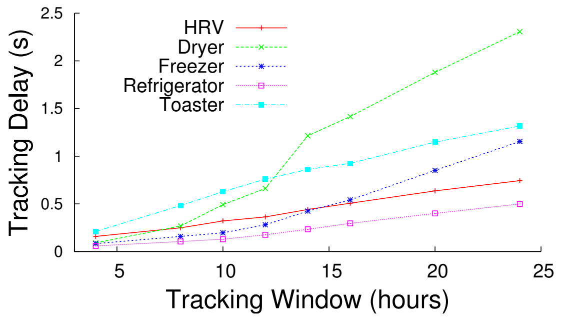


Figure 7. PowerPlay's tracking algorithm is efficient, with tracking delays of at most a few seconds.

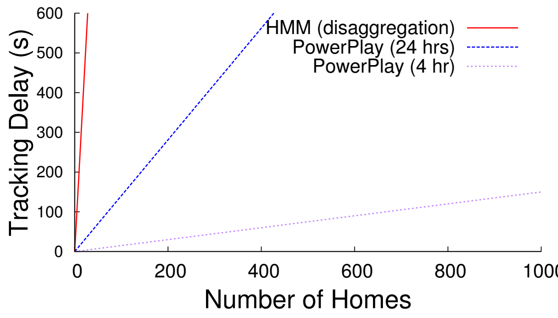


Figure 8. PowerPlay efficiency enables it to scale to many homes, while maintaining a low tracking delay.

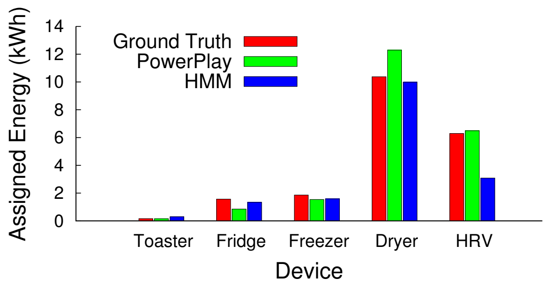


Figure 9. Both PowerPlay and the FHMM approach accurately assign the energy used by loads each day.

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

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. We quantify both PowerPlay's performance (with 24-hour and 4-hour tracking windows) and an FHMM approach following [18]. 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), the FHMM approach models each load as having only four power states and disaggregates at the level of circuits rather than individual loads. 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. Our example online FHMM incurs an 86 second tracking delay to track the loads in Figure 7 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 8, where the number of independent homes we track is on the *x*-axis (each home is an independent tracking process that runs in parallel). We see 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.*

Finally, we also consider PowerPlay's performance on embedded platforms that track a set of loads *within* a home, such as in an embedded energy monitoring and analytics platform [17]. 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. Tracking the same five loads as above in our deployment home with a 4-hour tracking window, PowerPlay achieves a tracking delay of 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.*

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. We take the conservative approach of training the FHMM on per-load data from the home that we disaggregate, although doing so is often not possible in practice, since disaggregation is typically only useful in homes where such training data is not available. As 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 9 shows the actual energy usage over an entire day for five loads, as well as the inferred energy usage from both 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. Our results are consistent with prior work on the FHMM approach, which performs as well, or better, than other prior approaches to

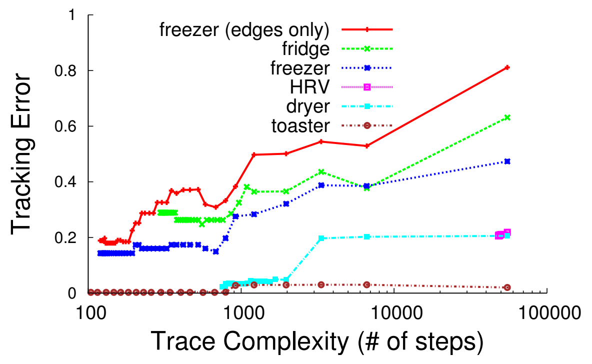


Figure 10. PowerPlay error factors when scaling up to highly noisy and complex smart meter data.

disaggregation [16, 18].

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.*

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 §2 to quantify per-load accuracy over time. In Figure 10, 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. By gradually adding circuits from our home deployment to the smart meter data. 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 circuit added to it. For each new circuit, we track the loads and compute the error factor per load on the new dataset. Figure 10 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.

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 11 then examines three specific points from the previous graph and compares them with the FHMM approach. In Figure 11(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 11(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.

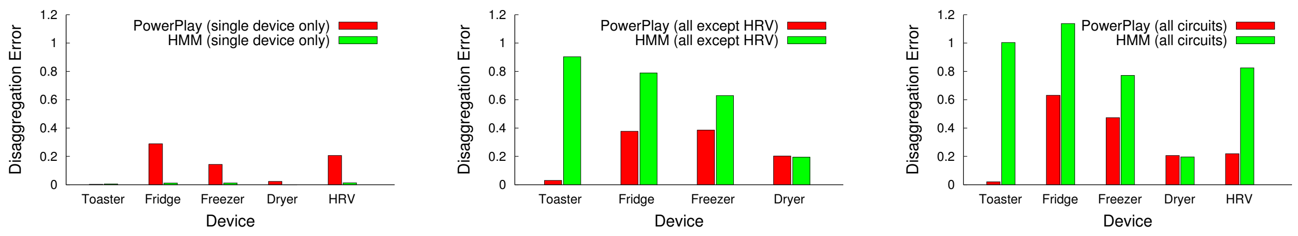
The result shows that PowerPlay is significantly more accurate than the FHMM approach for each load, with the exception of the clothes dryer. While PowerPlay is not more accurate than the FHMM approach at small scales, as in (a), with less "noisy" data, it is significantly more accurate as complexity increases. For example, PowerPlay is nearly perfect at detecting the second-to-second power usage of the toaster even within a highly complex trace, largely due to PowerPlay's highly accurate model of the toaster (as shown in Figure 11(a)). 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 ($\sim$6kW peak power versus $\sim$1kW peak power), such that the added complexity does not affect detection.

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.*

6.3 Case Study: Demand Response Capacity

Lastly, we consider a real application of scalable, online load tracking, where a utility wishes 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 they are consuming.

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. Finally, we use PowerPlay's output to query the set of active A/Cs



(a) Self Input (b) Aggregate Data minus HRV (c) Aggregate Data

Figure 11. PowerPlay is more robust to noisy smart meter data than the FHMM-based approach.

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 peak and 2.6kW 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,*

7 Conclusions

This paper 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. This paper enumerates an identifiable set of features common across loads, and then designs 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.

8 Acknowledgments

This research was supported by NSF grants CNS-1405826, CNS-1253063, CNS-1143655, CNS-0916577, and a grant from the Massachusetts Department of Energy Resources.

9 References

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

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

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

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

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

[6] Bidgely. `http://bidgely.com`.

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

[8] D. Chen, S. Barker, A. Subbaswamy, D. Irwin, and P. Shenoy. Non-Intrusive Occupancy Monitoring using Smart Meters. In *BuildSys*, November 2013.

[9] eGauge Energy Monitoring Solutions. `http://egauge.net`.

[10] 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`.

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

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

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

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

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

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

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

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

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

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

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

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

[23] Energy, Inc. `http://www.theenergydetective.com/`.

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

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

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