AutoPlug: An Automated Metadata Service for Smart Outlets

Lurdh Pradeep Reddy Ambati and David Irwin
University of Massachusetts Amherst

***Abstract*—Low-cost network-connected smart outlets are now available for monitoring, controlling, and scheduling the energy usage of electrical devices. As a result, such smart outlets are being integrated into automated home management systems, which remotely control them by analyzing and interpreting their data. However, to effectively interpret data and control devices, the system must know the type of device that is plugged into each smart outlet. Existing systems require users to manually input and maintain the outlet metadata that associates a device type with a smart outlet. Such manual operation is time-consuming and error-prone: users must initially inventory all outlet-to-device mappings, enter them into the management system, and then update this metadata every time a new device is plugged in or moves to a new outlet. Inaccurate metadata may cause systems to misinterpret data or issue incorrect control actions.**

To address the problem, we propose AutoPlug, a system that automatically identifies and tracks the devices plugged into smart outlets in real time without user intervention. AutoPlug combines machine learning techniques with time-series analysis of device energy data in real time to accurately identify and track devices on startup, and as they move from outlet-to-outlet. We show that AutoPlug achieves $\sim$90% identification accuracy on real data collected from 13 distinct device types, while also detecting when a device changes outlets with an accuracy >90%. We implement an AutoPlug prototype on a Raspberry Pi and deploy it live in a real home for a period of 20 days. We show that its performance enables it to monitor up to 25 outlets, while detecting new devices or changes in devices with <50s latency.

I. Introduction

The U.S. Energy Information Administration estimates that commercial and residential buildings account for 41% of U.S. energy usage, and over 75% of its electricity usage [1]. As a result, gathering detailed energy usage from buildings to optimize their energy consumption is critically important. Due to the high price of networked sensors, prior researchers have focused on analyzing power data from a single building-wide energy sensor to disaggregate it and estimate the energy usage of individual devices [2]. Unfortunately, such energy disaggregation, which is also known as Non-Intrusive Load Monitoring (NILM), is often highly inaccurate even in buildings with only a small number of devices [3]. However, recently, low-cost network-enabled energy sensors and switches have become widely available to consumers. The presence of these sensors can both aid in disaggregation or remove the need for it entirely. For example, many commercially-available smart power outlets cost <$50, including the Belkin WeMo [4], Insteon iMeter [5], and Z-Wave Smart Energy Switch [6]. In addition, research prototypes now exist that cost less than $20 [7]. These "smart" sensors and switches have the potential to enable deep visibility and control of the energy usage for each individual electrical device in a building. Ultimately, smart sensors and switches are the foundation of "smart" buildings that collect energy usage data from devices, combine it with external data on the environment, forecasts, energy prices, user occupanncy and comfort, etc., and analyze it to coordinate control of devices to optimize for energy usage, cost, user comfort, etc.

Smart energy sensors and switches may either be embedded into a device itself, or be attached externally to the device, e.g., as part of a power outlet. Embedding sensing and switching functions into devices enables users to perform a one-time association between a device's unique identifier, e.g., its MAC or other layer-2 address, and its building management system (BMS). While this association is often done manually, given well-defined standards, resource discovery protocols could also be developed to automate the device's initial configuration with the BMS. However, embedding such functions into devices is likely only feasible for devices that are large enough to warrant the additional complexity. The numerous miscellaneous electrical loads (MELs), which comprise a rapidly growing portion of building energy usage [8], are likely too small and inexpensive to warrant their own embedded sensing and control functions. In addition, existing appliances that do not have smart functions will continue to operate for many years. Further, this approach requires BMSs to interact over the network with untrusted devices that visitors may bring into the building, e.g., to register them with the BMS, which is a security concern both for the BMS and for visitors.

Thus, a more general approach is to separate the energy sensor/switch from the devices, often by embedding these functions into each building outlet. This approach requires instrumenting only a building's outlets, rather than its devices. As a result, the BMS need only be configured once based on the unique identifier associated with each outlet, and also its location (which is generally not available from device-level sensors). In addition, since the outlets are part of the building's administrative domain, they can be trusted by the BMS, alleviating it from interacting over the network with untrusted devices from visitors. However, such external sensing poses a significant metadata challenge: since the sensors are built into outlets, rather than devices, users must manually associate the outlet with the respective device that is plugged into it. Further, users must alter this device-to-outlet mapping every time devices are unplugged or move to a new outlet. While some devices, such as a refrigerator, rarely if ever move, other devices, such as laptops, frequently change outlets. Companies

978-1-5090-5117-5/16$31.00 ©2016 IEEE

typically provide smartphone or desktop apps to configure and monitor smart outlets, as well as schedule remote control of devices, e.g., to turn them on or off at specific times or based on custom triggers. These applications also provide basic energy data analytics, such as a device's energy consumption. The market for smart outlets and other home automation devices is expected to grow by 60% from 2012 to 2018 [9].

Energy data recorded by smart outlets is much less useful to a BMS if the data is not correctly associated with a device, as it prevents a BMS from providing an accurate per-device breakdown of energy usage and also may result in incorrect remote control actions, e.g., by switching the wrong devices on or off. The configuration of current applications for controlling smart outlets and collecting their energy data is manual, and typically based on the outlet and not the device. Thus, users can only view the energy usage of outlets or automate the control of specific outlets, and not devices. Providing such energy data and control for devices, regardless of the outlet they are plugged into, is more natural for users, as energy-efficiency optimizations are based on devices not outlets.

To address the problem, we design AutoPlug, an automated metadata service for smart outlets, which can automatically identify and track the devices plugged into smart outlets based on their energy data in real time. We present our system as a service, deployable in a wireless gateway that communicates with smart outlets, and has the ability to identify the appliance plugged into the outlets. This gateway maintains a record of both previously identified devices, as well as a real-time record of the smart outlet→device mappings. This gateway could be incorporated into modems like Google Onhub [10], or hubs like the Amazon Echo [11]. For example, the Amazon Echo can already communicate with Belkin Wemo, ZWave, and Zigbee sensors and switches. AutoPlug combines machine learning techniques along with analytical time-series models of device usage to accurately identify and track devices on startup, and as they move from outlet-to-outlet in real time.

Compared with prior work [12, 13, 14, 15], which focuses primarily on labeling devices based on their energy data offline using machine learning techniques, AutoPlug is designed to be a real-time system that identifies when a device moves from one outlet to another. Our basic approach is to combine time-series pattern matching techniques to recognize when the pattern of energy usage for a device changes, which indicates a new device has been plugged in. As we show, identifying such changes can also improve the offline machine learning techniques above by enabling them to accurately configure the time period over which they analyze data. We implement an AutoPlug prototype on a Raspberry Pi and deploy it live in a real home for a period of 20 days. Our results show that AutoPlug achieves $\sim$90% identification accuracy on real data collected from 13 distinct device types, and is also able to accurately detect when a device changes outlets with accuracy $>$90%. In addition, we show that AutoPlug is able to monitor up to 25 outlets on a Raspberry Pi 2, while detecting new devices or changes in devices with only a 50s latency.

II. Background

AutoPlug assumes a smart building that is equipped with smart outlets capable of recording and wirelessly transmitting their power consumption in real-time, e.g., at a 1Hz resolution, to a centralized gateway. The outlets may also be remotely controlled by the gateway, e.g., switched on or off. AutoPlug differs from prior work on identifying devices based on their energy data [12, 16] in that its goal is to operate online and identify the presence of new devices in real time, and track their movement from one outlet to another. Thus, in contrast to prior work, we evaluate not only AutoPlug's accuracy but also its performance in terms of its latency to identify devices.

Problem Statement. We define AutoPlug's outlet metadata problem as a combination of two distinct, but interlinked sub-problems. The first sub-problem is to identify the device D that is plugged into a smart outlet O_i over a period $[t_{start}, t_{end}]$, given time-series power data $P(t)$ from $[t_{start}, t_{end}]$. This problem is similar to the machine learning classification problem explored in prior work [12, 16], where the task is to map a given feature vector, which is based on processed time-series data, to a device label. As in prior work, AutoPlug processes the time-series power data to form a feature vector based on the data's statistical metrics. We then use well-known feature vectors from representative devices with known labels as training data to the classifier. After building the model, the classifier outputs a device's label based on an input feature vector. One notable difference between prior work and AutoPlug is the selection of the interval $[t_{start}, t_{end}]$ over which the classification occurs. Prior work generally performs this classification over a static time period, e.g., every 24 hours, which may result in inaccuracy if the device plugged into the outlet changes one or more times within the 24 hour period. In this case, the feature vector represents a variety of different features from multiple different devices. Instead, AutoPlug dynamically sets the interval based on the sub-problem below.

Our second sub-problem is to identify when a device is newly plugged into an outlet or changes from one outlet to another. MELs are often plugged into and out of outlets, especially in shared spaces such as living rooms or kitchens. We call this sub-problem "swap" detection using the same terminology from prior work, which first identified this problem [16]. However, prior work only applied the same classification techniques as above to detect such swaps. Unfortunately, the machine learning classification problem above is not well-suited to dynamically detecting such changes in outlets in real time, as these classifications are trained based on device features, rather than the "features" of a change. That is, they attempt to simply map features over a given time period to a single device label. Thus, prior approaches cannot accurately detect the presence of multiple devices over a time period. Given a smart outlet O_i and time series power data $P(t)$, swap detection is the problem of determining the time t_{change} when a new device is plugged into an outlet and is turned on. Swap detection has two key metrics: the accuracy of t_{change} and the latency to detect a change has occurred.

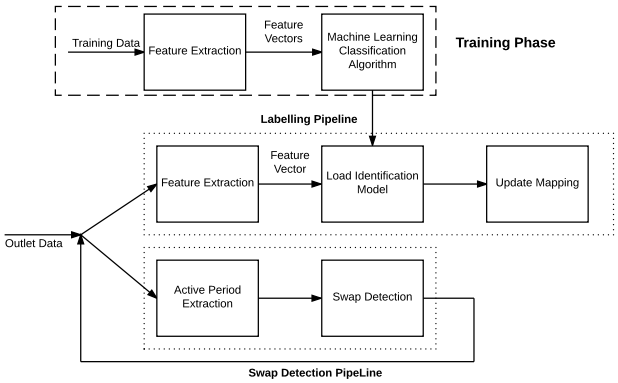


Fig. 1. AutoPlug Design Block Diagram

III. Design

We distill AutoPlug's two sub-problems of outlet metadata maintenance—device classification and swap detection—into the two design pipelines in Figure 1. The device classification sub-problem includes feature extraction from time-series power data, as a pre-processing step, followed by model building based on training data from existing device energy usage traces, and then load classification based on the learned model, which provides the output AutoPlug uses to update its device-to-outlet mapping, i.e., by modifying the database that stores the mapping. In contrast, the swap detection pipeline has only two stages: the active period extraction as a pre-processing step followed by time-series similarity matching. In active period extraction, AutoPlug divides the input time-series power data into distinct device active periods, which represent contiguous time periods where a device is active and consuming electricity. Note that if there is no energy consumption by an outlet, AutoPlug cannot determine whether a device is unplugged or whether it is simply not turned on.

A. Device Classification and Labeling

For device classification and labeling, similar to prior work, we first perform feature extraction by transforming a given window of time-series power data into a reduced set of statistical features, called a feature vector, that serves as input to a classifier. AutoPlug extracts features from both the raw data, as well as processed data consisting of a new time-series of energy deltas that represent the difference between consecutive power readings in the raw data. We use the latter time-series because changes in power are often more identifiable than the raw power level of a device. Below, we briefly review the specific features AutoPlug's classifier employs for model training and device identification. Note that these features are similar to features used in prior work [12, 16].

1) Statistical Features: We compute a simple set of statistical features for the two time-series above. Common features include the average, maximum, minimum, standard deviation and variance over each input time-series. These statistical features provide the classifier model characteristic and discriminative information for a specific device. In addition, we also compute an additional metric for our feature set: the number of energy deltas greater than a threshold value Δ_{OSC}. This metric gives insight into the dynamic behavior of the device's energy consumption, i.e., the frequency and magnitude of its variations in power, as shown in the equation below (where p_i is the average power of i^{th} outlet, and $\delta_>(x, y) = 1$ if $|y - x| > threshold$ and 0 otherwise).

$$\Delta_{OSC} = \sum_{i=2}^{N} \delta_>(p_i(t_i), p_i(t_i + 1)) \quad (1)$$

2) Duty Cycle: The duty cycle is the fraction of time a device has been active during a given window of time. This feature is useful in distinguishing continuously running devices from devices that run for shorter periods. The duty cycle feature indicates if an outlet's device is idle or active in the recent time-series window. We compute the duty cycle as the number of power readings greater than a threshold value divided by the total number of readings. This threshold value varies depending on the input time-series data.

3) Histogram Features: Devices also exhibit patterns of energy usage that are not captured by aggregate statistical metrics. Similar to prior work [12], to capture this, we separate the energy delta values of a device's time-series power data into separate bins of a histogram, which indirectly captures a device's energy usage pattern as a set of features amenable to classification. The selection of bin sizes is configurable, and affects the model's accuracy. We use 8 different overlapping bins spanning from 10W to 2500W. Each overlapping bin width is X to 5X, where X represents the starting power value for a bin. For example, our first bin is 10W-50W. For each bin, we calculate two features: a) a bin size, which represents the number of values that have populated the respective bin and b) an average time interval between the energy deltas in each bin. Thus, for 8 bins, there will be a total of 16 features that characterize the waveform of the time series data.

B. Detecting Outlet Changes

As discussed earlier, classification is not sufficient to accurately identify devices that change outlets in real time. In this case, the feature vector from an outlet's time-series energy data may represent a combination of two or more devices. The classifier, however, will provide only a single label, which may not match any of the devices plugged in, as the aggregate features above may significantly diverge from the individual features of any single device. Thus, detecting outlet changes is critical to the consistent maintenance of outlet metadata. Since standard classification is not well-suited to detect such real-time swaps, we design the detection technique below.

1) Active Period Extraction: First, to detect a device change in a smart outlet, we extract the active periods from the outlet's time-series power data. Each device alternates between active periods, where it consumes significant energy, and inactive periods, where it does not. Since some devices consume a small amount of standby, or vampire, power when inactive, we assume a device is inactive if its power usage falls below

a small threshold. Based on empirical data across a wide set of devices we set this threshold to 5W. The active period is then a continuous time period where the device operates over a power greater than this threshold. We delineate separate active periods if the inactive period is greater than a separation time threshold, e.g., one minute. That is, if there is an inactive period of greater than one minute we consider there to be two active periods before and after the inactive period. If the inactive period is less than one minute, we discard the inactive period and assume it was part of a brief lull in operation of a device's active period. Note that we tried more advanced techniques for extracting the active period, such as changepoint detection, but found that this simple technique performed similarly and was much more computationally efficient.

2) Time Series Matching: After we extract each active period from the input time-series power data in real time, we then compare it with the previous active period to determine if the device has changed outlets. In each case, AutoPlug signals a change in the device if the new active period is significantly different than the previous active period. We combine two different approaches to perform this comparison.

Time-series Distance. There are multiple functions available to compute the distance between two time-series, such as Euclidean distance or Dynamic Time Warping (DTW) [17]. DTW improves on Euclidean distance, as it is less sensitive to slight differences in the alignment and shape of the time-series pattern, i.e., it is able to slightly "warp" each time-series to better align them and reduce the distance. Thus, DTW is robust to data sequences of different lengths unlike with Euclidean distance [17], as traces are "warped" non-linearly in the time dimension to compute a measure of their similarity. However, the DTW algorithm is expensive, as it has $O(n^2)$ time complexity, where n is length of longest data sequence. Thus, the longer the sequence in length, the more time it takes for AutoPlug to compute the DTW distance, which may not scale well on embedded devices like a Raspberry Pi or Arduino, commonly used as gateway devices. As we show in our experiments, we coarsen our data (from 1Hz resolution to 0.2 Hz resolution) before applying DTW to improve performance. Thus, in this approach, we compute the DTW distance between two consecutive active periods and signal a change when it exceeds a specified threshold.

Curve Fitting. Another approach is to fit a function to the data, e.g., such as a logarithmic growth function, and then compare the parameters of the best fit function for both active periods. In this case, we signal a change if the percentage difference between the parameters exceeds a specified threshold. Curve fitting is a method to construct the best fit of a mathematical function for the input data sequence, given the curve type or reference mathematical distribution. In this approach, we compute the parameters of the best fit logarithmic growth function to the active period, as prior work shows that this function approximates the energy usage pattern on startup for a wide range of devices [18].

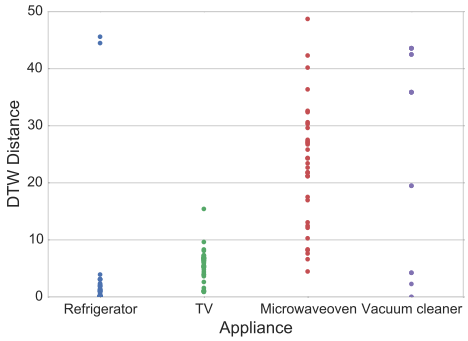


Fig. 2. Strip plot of DTW distances between sequences of the same appliance, broken down by appliance

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

Using the logarithmic growth function, curve fitting on a given data set computes two parameters p_{base} and λ. p_{base} is the starting power level of the best curve fit and λ is growth parameter. In our approach, we compute parameters for both the active periods and then we compare the respective p_{base} parameters, and finally compute similarity S as the percentage difference in p_{base} of the both, where p_{base1} and p_{base2} are parameters for the active periods, respectively.

$$S = \frac{|p_{base1} - p_{base2}|}{max(p_{base1}, p_{base2})} * 100 \quad (2)$$

Approach Selection. We use the two approaches above in different circumstances. Specifically, if the length of an active period is short, e.g., less than three minutes in our experiments, then we compare two sequences using the second approach, since the logarithmic growth characteristic of many devices is generally short-lived. In contrast, if the length of the active period is long, e.g., greater than three minutes, we use DTW, as longer active periods tend to exhibit more variations in power usage that do not permit a single curve fitting.

AutoPlug signals a change if the similarity score or DTW distance exceeds a threshold. As an example, we measure the DTW distance between two active periods for four device types and illustrate the results in a strip-plot in Figure 2. The figure shows that the DTW distance for the refrigerator and TV are well below 10 (with few exceptions), but that the microwave and vacuum have DTW distances scattered in the range of 0 to 50. Thus, selecting the DTW threshold for the microwave and vacuum is more difficult than for the refrigerator and TV. However, this is due largely to the shorter operating cycle of the microwave and vacuum, which in this case is below our threshold of three minutes. Thus, AutoPlug uses curve fitting for these shorter active periods, as the DTW distance threshold is more variable for these periods.

C. Window Size and Update Frequency

AutoPlug adapts the data window size and frequency at which it runs the classification problem above. Prior work [12]

uses a static window size of 24 hours and updated the classification offline once per day. Instead, AutoPlug sets the window size and update interval dynamically when it detects a change in the outlet. That is, the window size for the classification of an outlet starts from the last change detected to the current time. In addition, after a change AutoPlug periodically re-runs the classification, as the classification accuracy increases as more data is collected after a device swap. The period at which AutoPlug re-runs the classification based on new data is frequent, e.g., every 15 minutes, as new data significantly improves classification immediately after an outlet change. AutoPlug stops re-running the classification when the confidence in the labeling both reaches a specified threshold and does not significantly improve with new data. Note that this approach results in AutoPlug potentially mislabeling a device immediately after a change, as there is not much data, and then correcting itself as it collects more data.

IV. Implementation

We have implemented Autoplug in python using the Scikit-learn [19] and Scipy [20] stack. Scikit-learn is an open-source machine learning library for python, which has a collection of classification, regression and clustering algorithms. SciPy has a collection of powerful scientific computing libraries for data processing and visualization, as well as modules for performing curve fitting. We use the implementation of Dynamic Time Warping from a standard machine learning library for python. AutoPlug maintains a simple database table where each row stores a device label, an outlet label, a start time for the association, and the duration of the association.

We have implemented an AutoPlug prototype system on a Raspberry Pi 2, and deployed it in a real home. We instrument the home with four Belkin Wemo Insight Switches [4]. The Belkin Wemo Insight Switches are WiFi-enabled smart outlets, which are programmatically accessible via the ouimeaux python API [21]. The Raspberry Pi acts as a gateway to gather outlet power data, using the `ouimeaux` API to poll each outlet for its energy usage at a 1Hz resolution, and implement AutoPlug's techniques from the previous section.

For AutoPlug's initial training for classification, we use device-level data from both the Tracebase public data set [22] and data collected from a real home instrumented with eGauges [23]. Tracebase includes 1Hz data resolution and derives from 158 devices at different homes over a span of 158 days in Germany in 2012. The eGauge is a standard network-connected power meter, which makes its data available through a programmatic API. We train our classifier on 13 different device types from these data sets, including a coffee maker, dishwasher, freezer, lamp, clothes dryer, microwave, printer, refrigerator, toaster, TV, washing machine, vacuum cleaner and laptop charger. In addition to the real data above, we also generate synthetic building datasets using Tracebase and eGauge by replaying data from multiple devices, as if they were present in a single home. Our virtual datasets assume the refrigerator, TV, and microwave are static and do not change outlets, while devices, such as a lamp, laptop, and vacuum cleaner, potentially do change outlets.

Classifier	Accuracy(%)	Training-Time(sec)
Naïve Bayes	69.23	0.165
Support Vector Machine	76.60	260.23
Random Forest Classifier	89.94	1.670

TABLE I
ACCURACY OF DIFFERENT CLASSIFIERS ON OUR DATASET.

V. Evaluation

We evaluate AutoPlug's accuracy on our multiple public data sets, on data collected during the live deployment, and on the virtual datasets we construct from real data. In addition, we also evaluate its performance, in terms of the average computation time per update, i.e. latency, on multiple platforms, including a Raspberry Pi and Macmini.

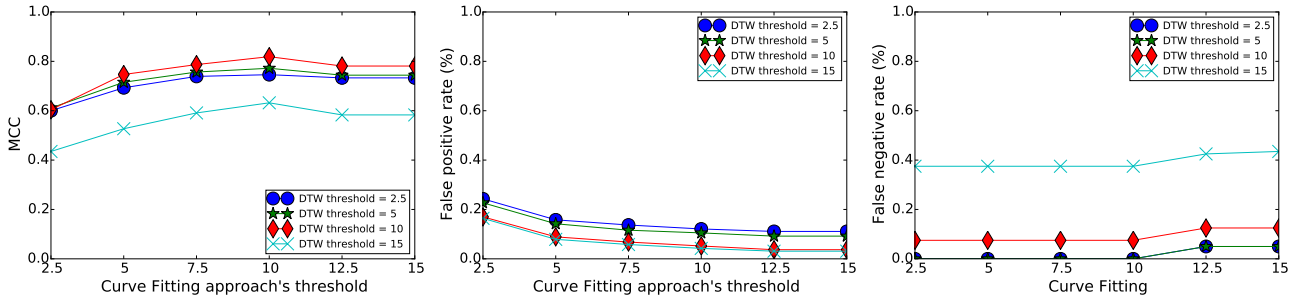
A. Accuracy

AutoPlug operates in an online fashion by continuously polling each smart outlet every second, such that it updates the smart outlet's label at a dynamic interval based on the detection of a swap. We evaluate both classification accuracy and swap detection accuracy. As we discuss, there is a trade-off between the latency of detecting a swap and its accuracy.

1) Classification Accuracy: We divide our dataset into blocks of two hours each, and then we transform each data block into a feature vector as described previously, which we use for initial classification. Similar to prior work, we consider two scenarios of evaluation for the classification: identifying previously observed (or "seen") devices and previously unobserved (or "unseen") devices. Identifying seen devices is the case where respective device data is already in our training data, where with unseen devices the classifier predicts the label of a device whose data is not in the training data.

Identifying Seen Devices: We evaluate AutoPlug's device identification accuracy when we train and test the classifier on the complete dataset. We use this experiment to evaluate the accuracy of multiple classifiers, including Random Forest [24], Support Vector Machine, and Naïve Bayes. We perform 5-fold cross validation on the complete dataset to measure the model accuracy of above classifiers. Table I shows that the accuracy is highest for the Random Forest Classifier. Its accuracy of 90% is similar to the 93% accuracy presented in prior work on NILI, which addresses a similar problem [12]. However, NILI only works on a dataset size that includes 24 hours of energy data, which results in more information in each instance. In addition, NILI only updates each outlet's label once per day, while our approach performs an update at least once every 2 hours, and is dynamically determined. Table I is for static devices that do not change outlets. If we insert a change in outlet for each device, AutoPlug with the Random Forest Classifier maintains an accuracy of 89%, while NILI accuracy drops to 79%.

Identifying Unseen Devices: In general, training data for devices is not known *a priori*, and thus identifying previously unobserved devices is also important. For this experiment, based on our results above, we train the Random Forest



(a) MCC (b) False positive rate (c) False negative rate

Fig. 4. Accuracy of AutoPlug's time-series-based technique for detecting changes to outlets.

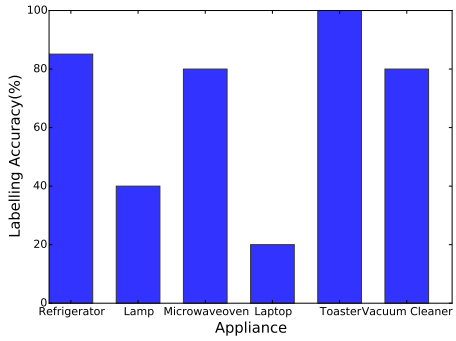


Fig. 3. Detailed identification evaluation per device for devices not in the training set with the Random Forest Classifier

Classifier on the complete dataset and then compute testing accuracy for the data collected during the live deployment, which includes devices not in the training data. Note that the live deployment also included devices changing outlets. We observe that overall AutoPlug accuracy in this live deployment is close to 76% where the AutoPlug correctly labels 236 instances out of 310 instances. In comparison, the NILI approach that assumes devices never change outlets yielded an accuracy of only 64%. Figure 3 shows the per device break down of the accuracy. Some devices exhibit worse accuracy than others. For example, the labeling for the lamp is less accurate because, in the live deployment, the lamp's power output was 70W, while in the training data set the lamp data in some cases was 120W and higher. As a result, the Random Forest Classifier fails to classify the 70W lamp correctly. Thus, we expect these results to improve with more training data.

2) Detecting Outlet Changes: Since the output of swap detection is a binary classification, e.g., either a swap is detected or it is not, we evaluate its accuracy based on the false positive rate, false negative rate and the Matthews Correlation Coefficient (MCC) [25]. A false positive represents the number of instances in which AutoPlug detects a swap that is incorrect. Similarly, false negatives represent instances where the AutoPlug fails to detect a swap. Similarly, the MCC is a quality measure of binary classification that takes both the false positives and false negatives into account, and represents a balanced measure of a binary classifier's performance. The MCC's value is between -1 and $+1$, where $+1$ indicates a perfect prediction, 0 indicates a random prediction and -1 indicates a total mismatch of observations and predictions.

To evaluate swap detection, we compute the metrics above for our live deployment data and virtual data sets assembled from the Tracebase [22] and eGauge [23]. During the live deployment, we plug in six different devices into smart outlets, such that three of them operate in the living room and the other three operate in the kitchen. We then swap one device with other that operates in the same room.

Our false positive rate and false negative rate will depend on the threshold values for swap detection. To find the optimal threshold value for both of our swap detection approaches, we consider our DTW approach's threshold from 2.5 to 15 and our curve fitting approach's threshold from 2.5 to 15. We then compute the MCC, false positive, and false negative rate for each set of threshold values. Figure 4(b), and (c) shows that the false positive rate increases as the threshold values decrease, while the false negative rate decreases as the threshold values decrease. Thus, a threshold combination of 10 and 10 for both the approaches respectively is a suitable choice as this keeps the false positive rate and false negative rate relatively low and maximizes the MCC value, shown in (a). As the figure shows, the false positive and false negative rates of AutoPlug with optimal threshold values are 5.2% and 7.5%, which translates to 92% swap detection rate.

Figure 4(a) shows that AutoPlug's MCC for swap detection is near 0.8. We also compared this with an approach that detects changes solely based on classification, as in [16], and not detecting changes in the pattern of time-series data. For this classification-based approach, we compare the output classification label of an outlet from the present data window to previous data window. If the labels do not match then we signal that a device has changed outlets; if they do match, then we signal a device has not changed outlets. This classification-based approach yielded an MCC of 0.31 and a false positive rate of 16%. This approach differs from AutoPlug in that the classification requires much more data to accurately re-classify a device, while our time-series-based approach is able to re-classify as soon as it recognizes that the pattern does not match the data pattern. Since such classification is done over data features that are aggregate statistical metrics of the data, more data results in a higher classification accuracy.

3) Dynamically Setting the Window Size: Recall that, unlike prior work, AutoPlug dynamically sets its window size and

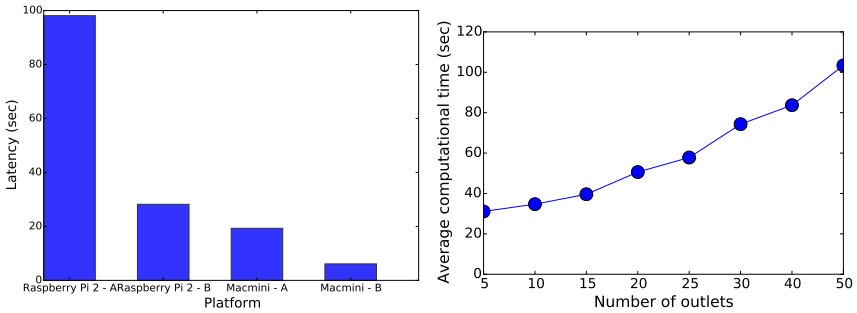


Fig. 5. AutoPlug's overhead for different platforms and configurations (a), and its overhead for varying numbers of outlets (b).

update frequency for classification based on its swap detection. AutoPlug automatically starts a new window after detecting a swap and then continuously updates the classification as new data becomes available. AutoPlug stops the classification when the confidence level for the device reaches a threshold and stops improving. The confidence level is the probability assigned to the true label from the probability distribution over the set of labels considered by the classifier. If the confidence level is over 50%, then it indicates that the classifier output is likely correct, and less likely if under 50%.

We compute the confidence levels for four devices: a lamp, laptop, microwave oven and refrigerator over three data window sizes (0.5 hours, 1 hour, and 2 hours) after a change. We then compute confidence levels over multiple active periods of the device and average those values, respectively. Figure 6 shows the results, which indicate that the microwave oven can be clearly labeled in 0.5 hours, but refrigerator and laptop need more time for accurate detection. Note in the laptop and refrigerator cases, when data window size was 0.5 hours, the classifier failed to assign their correct label, but AutoPlug corrects itself once more data becomes available.

B. Performance

We next measure the average computation time per update i.e. latency, to evaluate AutoPlug's performance. Since AutoPlug stores only recent two hour power data per outlet, I/O is negligible, and computation time will depend on a) the number of outlets being monitored and b) the computational overhead of the swap detection process, especially the DTW approach given its high overhead. Here, we configure AutoPlug in two configurations—A and B—such that A is the default configuration and B adds a processing step that reduces the resolution of the active periods by a factor of five before applying DTW, e.g., from 1Hz resolution to 0.2Hz resolution. Note that, we have evaluated accuracy of system in configuration B.

We first investigate how the latency varies for AutoPlug running on different platforms. For this experiment we implement AutoPlug on a Raspberry Pi 2 running Ubuntu Linux, as well as a Macmini running OSX. From Figure 5(a) we see that the latency is modest in the case of Macmini, but increases for Raspberry Pi 2, particularly when AutoPlug is in configuration A, due to the computational overhead of DTW. Next, we measure the latency of the AutoPlug for varying number of smart outlets. We conduct this experiment with

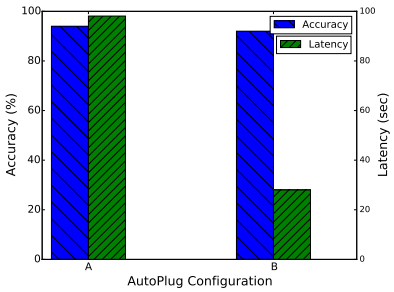


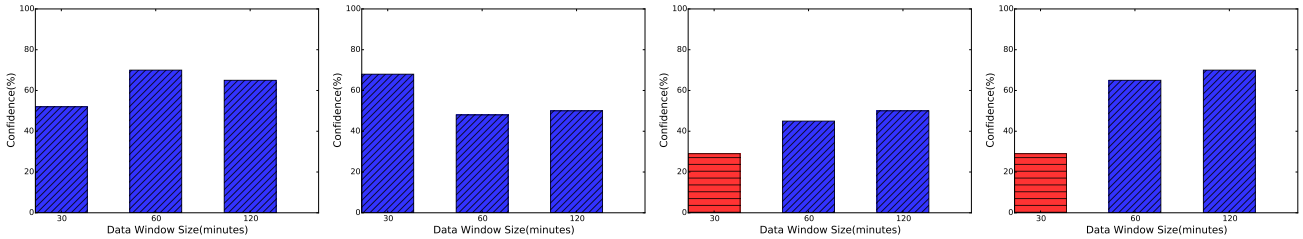
Fig. 7. Accuracy and latency for two AutoPlug configurations.

AutoPlug in simulation (using configuration B) implemented on a Raspberry Pi 2. Also, for this experiment we assume that all the outlets are active. From Figure 5(b), we see that the latency increases as the number of outlets increases. When the number of active outlets increases beyond 25, there is a sharp increase in latency, due to the CPU nearing saturation.

Finally, Figure 7 shows the trade-off between AutoPlug's latency and swap detection accuracy. Here, we configure AutoPlug on a Raspberry Pi 2 monitoring four smart outlets. We see that latency varies by a large margin from configuration A to B, but accuracy in both the configurations is above 92% with configuration A being marginally higher, as expected due to the higher data resolution. However, the increase in accuracy over configuration B, which reduces the resolution of the data to reduce DTW overhead, is not significant, indicating the benefit of this optimization.

VI. Related Work

While prior work [12, 13, 14, 15] has investigated classification techniques for device identification from energy data it does it offline at a static interval and does not i) adaptively determine the interval over which to classify devices, which decreases its accuracy if devices change outlets, and ii) detect and report in real time when such a device change occurs. AutoPlug applies time-series techniques to detect such changes in real time, and then feeds this information back to the classifier to determine when to re-classify an outlet (and the interval of classification). AutoPlug is able to quickly recognize outlet changes with little data, and feed these times back to the classifier to determine the period over which to classify data. Thus, AutoPlug partially focuses on system performance and the latency of detection, in addition to



(a) Lamp (b) Microwave oven (c) Laptop (d) Refrigerator

Fig. 6. Confidence Level versus data window size for different devices.

classification accuracy, while the prior work above focuses only on accuracy. A. Leonardi et al. [16] also take a similar approach for labeling smart outlets using classification, and extend it to detect changes in outlets. We show that AutoPlug outperforms such a classification-based approach in Section V.A.

Related work also focused on analyzing existing metadata in BMSs to discern the types of devices, largely in the context of commercial buildings [26]. This work differs from our work in that the labels in the BMSs already exist, but their encoding is not known, e.g., due to ad hoc encoding rules for labels being manually entered by an operator, and must be derived automatically. The work above has led to efforts to standardize naming conventions for metadata in commercial buildings. AutoPlug does not focus on how to name the devices once identified, and could use such standardized naming conventions.

VII. Conclusion

In this paper, we present AutoPlug, a system for identifying and tracking devices plugged into smart outlets in real time. Similar to prior work, AutoPlug transforms the given input time-series energy data into a compact set of features and then uses an off-the-shelf classifier to identify the loads. However, AutoPlug employs time-series pattern matching, e.g., using DTW and curve fitting, to detect changes in devices in real time and to track loads as they move from one outlet to another. AutoPlug is then able to feed this information back to the classifier to determine when to re-classify a device and over what time period. Our results show that we can achieve $\sim$90% load identification accuracy using the data from the 13 device types collected from various real homes and is also able to detect device changes with accuracy $>$90%. Further, our results show that our AutoPlug prototype deployed live on a Raspberry Pi is able to monitor as many as 25 outlets and support a 50s identification latency.

Acknowledgements. This research is supported by NSF grants IIP-1534080, CNS-1405826, CNS-1253063, CNS-1505422, and the Massachusetts Department of Energy Resources.

References

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

[2] G. Hart, "Nonintrusive Appliance Load Monitoring," *Proceedings of the IEEE*, vol. 80, no. 12, December 19 92.

[3] K. Armel, A. Gupta, G. Shrimali, and A. Albert, "Is Disaggregation the Holy Grail of Energy Efficiency? the Case of Electricity," *Energy Policy*, vol. 52, no. 1, January 2013.

[4] "Belkin Wemo Insight Switch," http://www.belkin.com/us/p/P-F7C029/, February 2016.

[5] iMeter Solo, http://www.insteon.net/2423A1-iMeter-Solo.html.

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

[7] S. DeBruin, B. Ghena, Y. Kuo, and P. Dutta, "PowerBlade: A Low-Profile, True-Power, Plug-Through Energy Meter," in *SenSys*, November 2015.

[8] "Analysis and Representation of Miscellaneous Electric Loads in NEMS," U.S. Energy Information Administration, Tech. Rep., December 2013.

[9] M. Brennan, "House of the Future: How Automation Tech is Transforming Homes," *Forbes*, October 2013.

[10] G. OnHub, https://on.google.com/hub/.

[11] A. Alexa, http://www.amazon.com/Amazon-Echo-Bluetooth-Speaker-with-WiFi-Alexa/dp/B00X4WHP5E.

[12] S. Barker, M. Mushtag, D. Irwin, and P. Shenoy, "Non-Intrusive Load Identification for Smart Outlets," in *SmartGridComm*, November 2014.

[13] A. Ridi, C. Gisler, and J.Hennebert, "Automatic Identification of Electrical Appliances using Smart Plugs," in *WoSSPA*, May 2013.

[14] D. Zuffrey, C. Gisler, A. Khaled, and J. Hennebert, "Machine Learning Approaches for Electric Appliance Identification," in *ISSPA*, July 2012.

[15] J. Gao, J. Ploennigs, and M. Berges, "A Data-driven Meta-data Inference Framework for Building Automation Systems," in *BuildSys*, November 2015.

[16] A. Leonardi, H. Ziekow, and A. R. D. Konchalenkov, "Detecting Smart Plug Configuration Changes in Smart Homes," in *Smart SysTech*, July 2014.

[17] C. Ratanamahatana and E. Keogh, "Three Myths about Dynamic Time Warping," in *ICDM*, April 2005.

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

[19] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, 2011.

[20] I. M. et al, "ouimeaux: Open Source Control for Belkin Wemo Devices," 2013. [Online]. Available: http://ouimeaux.readthedocs.io/en/stable/readme.html

[21] E. Jones, T. Oliphant, P. Peterson *et al.*, "SciPy: Open Source Scientific Tools for Python," 2001. [Online]. Available: http://www.scipy.org/

[22] A. Reinhardt, P. Baumann, D. Burgstahler, M. Hollick, H. Chonov, M. Werner, and R. Steinmetz, "On the Accuracy of Appliance Identification Based on Distributed Load Metering Data," in *SustainIT*, October 2012.

[23] "eGauge Energy Monitoring Solutions," http://www.egauge.net/, 2013.

[24] A. Liaw and M. Wiener, "Classification and Regression by Random Forest," *R News*, vol. 2, no. 3, 2002.

[25] P. Baldi, S. Brunak, Y. Chauvin, C. A. F. Andersen, and H. Nielsen, "Assessing the Accuracy of Prediction Algorithms for Classification: An Overview," *Bioinformatics*, vol. 16, no. 5, 2000.

[26] A. Bhattacharya, D. Hong, D. Culler, J. Ortiz, K. Whitehouse, and E. Wu, "Automated Metadata Construction To Support Portable Building Applications," in *BuildSys*, November 2015.