Distributed Non-Intrusive Load Monitoring

David C. Bergman, Dong Jin, Joshua P. Juen, Naoki Tanaka, Carl A. Gunter
University of Illinois at Urbana-Champaign
{dcbergma, dongjin2, juen1, tanaka5, cgunter}@illinois.edu

Andrew K. Wright
N-Dimension
andrew.wright@n-dimension.com

***Abstract*—Smart grid support for demand response provides strategies for an electricity service provider to shed loads during peak usage periods with minimal consumer inconvenience. Direct load control is a strategy for doing this in which consumers enroll appliances such as electric water heaters, air conditioners, and battery vehicles in a program to respond to load shed instructions in exchange for a discount on electricity prices or other incentive. The effectiveness of direct control depends on the ability of the provider to verify that appliances respond to load shed instructions. A technique called non-intrusive load monitoring, in which electric power meter readings are used to identify loads generated by specific appliances, provides a practical strategy for load shed verification in residences. Non-intrusive load shed verification (NLSV) promises to greatly simplify trust assumptions required for the deployment of direct control.**

Implementing NLSV on resource constrained smart meters is problematic. This paper describes and evaluates a distributed non-intrusive load monitoring algorithm that is split between a capable backend system and a typical smart meter in the field. [1]

I. Introduction

Non-Intrusive Load Monitoring (NILM) [1] is a technique for analyzing aggregate electrical load data together with appliance profile data to decompose the aggregate load into a family of appliance loads that explain it. NILM algorithms have developed significantly [2]–[13] since their introduction, and current algorithms are able to provide accurate decompositions in many practical cases. However, these advanced techniques rely on significant computational capabilities. This leaves a role for *distributed NILM*, where the algorithms can be split into two parts, one that runs on a capable backend system and another that can be run on a typical residential advanced meter. For example, the genetic algorithms used to eliminate the interactive training phase take ten minutes to run on a 3GHz Pentium 4 machine while the typical AMI meter is running at below a hundred megahertz. Advanced techniques such as the frequency analysis require meter readings at the millisecond scale [6]–[13] while AMI meters typically provide no better than one reading per second. Moreover, NILM techniques work best on large data sets while AMI meters have little memory and AMI networks do not have large bandwidth.

There are at least two potential uses for dNILM: (1) demand response verification [14] and (2) user recommendations. As for (1), in order to provide a stable electrical system, load must not exceed what can be provided. Various tactics exist to keep this balance in check. For instance, an Energy Service Provider (ESP) may offer discounts to users that reduce their load whenever the ESP sends a request. Without having to authenticate with each individual appliance, dNILM can instead allow the ESP to observe compliance through the trusted meter with only a limited amount of data being collected by the utility. As for (2), after an appliance profile is generated for a household, it can be compared to other households' appliance profiles. By observing similar households, it may be possible to give recommendations to the users. For instance, one family of four may be interested in knowing how its phantom load total compares to the average family of four's phantom load total. Again, with dNILM this can be done with only limited amounts of data being collected by the utility.

The goal of this paper is to describe and analyze a dNILM algorithm. The main insight is that by utilizing a network of semi-capable meters, we can distribute the NILM workload and avoid transfering large amounts of data in the process. The contribution of this work is, therefore, the ability to distribute the NILM workload and gain performance while enduring only a small accuracy penalty.

The paper is divided into two primary parts, one devoted to the description of our dNILM algorithm and the other to its analysis using data collected from a typical residence.

II. A Distributed NILM Algorithm

A. Overview

The goal of dNILM is to utilize computationally limited meters to analyze as much data as possible while still allowing advanced NILM techniques to be employed at the backend. The system must carefully limit the amount of bandwidth required between the controller at the backend and the meters, and also efficiently utilizing the processing power required on both while still maintaining accuracy. Thus the fully functional NILM algorithm must be designed across the meters and the controller.

The first step in typical NILM is the scanning for discrete step events, a process which is not computationally expensive. By only requiring the meters to log edge events, the amount of recorded data is reduced by two orders of magnitude. Furthermore, with an accurate table of load characteristics, a simple matching algorithm can be employed to identify large loads. These observations led to the following scheme composed of two phases. In the normal operating phase, the

[1]David C. Bergman, Dong Jin, Joshua Juen, Naoki Tanaka, Carl Gunter and Andrew Wright. Distributed Non-Intrusive Load Monitoring. In Proceedings of the IEEE/PES Conference on Innovative Smart Grid Technologies (ISGT 2011), Anaheim, CA, USA, January 2011

meters will take readings and conduct real time edge detection. Upon encountering an edge, they will parse a static table of load signatures and use a matching algorithm in an attempt to detect what loads are currently operating. The meter will then update a dynamic table to keep track of what appliances are currently running. By keeping this table up to date, the meters will always be prepared to respond to status requests from the backend controller. The learning phase is designed to identify which loads exist in the meter's environment. During this phase, the meters will collect data and parse it to produce edge events, but will then transmit them back to the backend. The backend will then collect data over a period of time. The backend will then conduct more advanced computationally intensive NILM techniques – namely, cluster analysis and finite state machine creation and optimization – in order to build the static state table to be used in the monitoring phase on the meter. Thus, more advanced techniques will be leveraged in ways that overcome hardware limitations.

Our algorithm implements a simple clustering analysis of the events and then develops a static state table using a genetic algorithm to optimize a set of possible finite state machines. In order for this scheme to be viable, it is necessary to confirm that the sensor data can be compressed reasonably enough to send over limited bandwidth links, that the sensors can identify loads based off of an accurate static table, and that a reasonable learning phase will allow the controller to build an accurate static table for the sensors. Ideally, the sensors will be able to identify the same loads running with limited learning as the fully functional NILM on the controller.

B. Event Detection

As a part of both the learning phase and the monitoring phase, the first goal of the meter is to detect abrupt changes in the power readings which correspond to loads changing state. Thus, the event detector produces an output whenever there is sufficient change from the current power value and by ignoring minor changes, meter readings collected every second can be represented in a compressed form.

The input is a series of power data tuples. Depending on the source, this tuple typically includes the following: $\{$time stamp, real power value, apparent power value, reactive power value$\}$. For robustness, the algorithms are also written to handle sources that do not measure reactive power. Hence, sometimes our tuple is represented as $\{$time stamp, real power value$\}$. In either case, the algorithm uses a series of tuples as the input and creates a series of tuples with the same format for the output. These output tuples represent sudden changes in power, produced when an appliance changes state.

The algorithm operates by keeping a running average of power values for a window of data. Every time a new event is detected, represented by a change above a predefined threshold, the current running average is output as an event corresponding to the time from the beginning of that period. Next, the running average is reset and the algorithm moves on to the next period. The algorithm also has a notion of stability. That is, a value has to be relatively constant for a short period of time before an event is said to have occurred. In this manner, transient states are ignored as events.

C. The Learning Phase

Power changes must be identified in order to classify what large appliances are present and group each power change with the respective load. Once the loads are identified, they are placed in a static table and returned to the meters to be used during the real-time phase. The most basic power model is the on/off model; however, not all loads can be explained by just on/off states. To accurately identify present loads, we propose an algorithm based upon genetic algorithms and dynamic programming [3], [4] that builds finite state machines(FSM) from clusters of edge events. Our implementation is based on the Java Genetic Algorithm Package (JGAP) [15]. The algorithm takes as input the power change clusters and edge events and generates the static appliance table by calculating the most likely combination of FSM to fit the input data.

The overall workflow of the algorithm is illustrated in Figure 1 (derived from Baranski's algorithm [3] with some modifications). Each step is explained in the following sections.

1) Clustering Algorithm: The first step to analyze the data on the backend is to establish clusters of on and off events in order to begin identifying what appliances are being seen by the meters. The clustering algorithm accomplishes this by taking an interval of data and grouping like events by their respective power changes.

The algorithm retrieves the first element of the event list, determines the power change, and then searches the rest of the array for matching events by assigning the first event to a new cluster. Each time an event is found that matches the current cluster, it is removed from the event list and placed in a list for that cluster item. If the event is positive, then the algorithm has found another on event. If the event is negative, then it has found an off event. The power value for the current cluster is then averaged again and used to search the list. That way, the list is searched for the current average of the events in the cluster. Lastly, the algorithm computes the final mean and standard deviation for use on the meter and in the appliance table building algorithm. The algorithm also returns how many times each cluster event was turned on or off during the collection period.

2) Building the State Machine: The first step to building a finite state machine is to select clusters that represent the states of a load. For N_c clusters, there are 2^{N_c} possible combinations of clusters to select. Each combination can be a candidate for the possible states of a load.

Because the number of combinations becomes huge when N_c is large, it is impossible to examine all possible combinations. Thus, a genetic algorithm is utilized to select only promising combinations of clusters. The genetic algorithm creates a matrix X that has N_c columns where each column corresponds to a cluster and each row corresponds to a candidate appliance. Each element in the matrix is binary, i.e., 0 or 1. If it is 1, then the corresponding cluster is considered to be one of the states of a load. The maximum number of

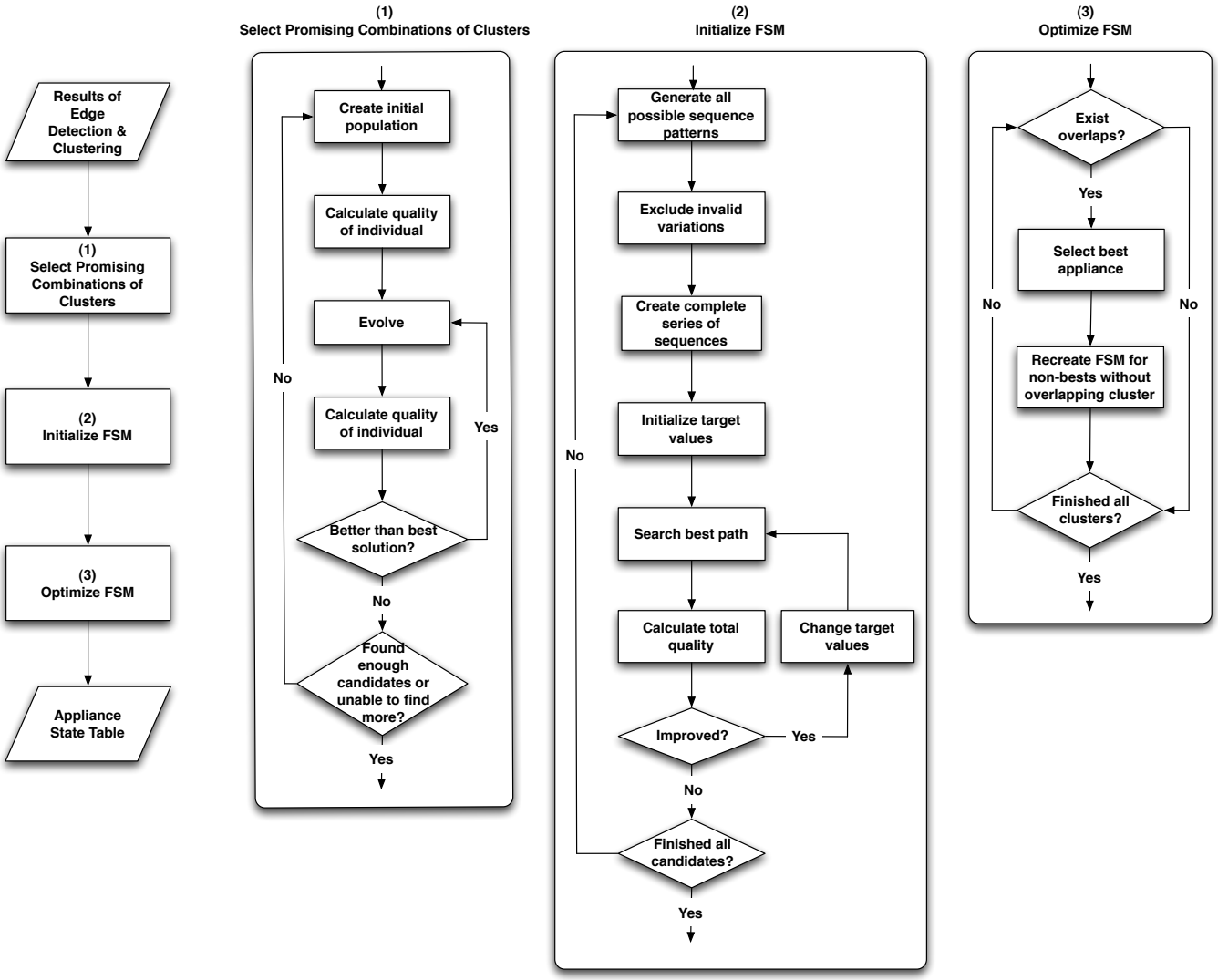

Fig. 1. Algorithm for Building Appliance Static State Table

rows is configurable. The genetic algorithm repeats until it finds the maximum number of combinations or it fails to find more promising candidates. The quality of each row of X is evaluated based on how close the sum of power changes is to zero. This is done because the state transitions of any load should start and end with an off state.

3) Initialization of the Finite State Machines: Next, every row of X is used to initialize a finite state machine. In order to create a finite state machine, the sequence of power changes must be determined. Because the number of permutations becomes huge when the number of selected clusters in a row of X is large, the program puts a limit, which is configurable, on the number of states for a single load. One example used in other work [3] is the combination of 100W, 50W and -150W. There are 3! = 6 possible variations of sequence patterns for this combination, but not all variations are valid because some variations result in a negative power value in the middle of the sequence which is unacceptable. For example, if the sequence of power changes is 100W, -150W, and 50W, then the power value becomes -50W after the second power change. Thus, such variations are excluded from consideration. Similarly, we also exclude variations that result in 0W in the middle of their state transitions or that do not come back to 0W in their last transitions. The algorithm then selects the best variation from the valid sequence patterns based on the frequency with which each sequence pattern occurs in the observed switch events and the quality criterion from Baransky [3]. Here, all observed switch events included in all the clusters (represented in each row of X) are used as power values of the states. Additionally, the series of the sequences that maximize the quality value are searched using a dynamic programming approach. We use two properties from Baransky [3] for the calculation of quality: (1) the time duration between state changes in a sequence, and (2) the deviation between the observed power value and the corresponding value of the cluster (i.e., the average of all power values in the cluster). The target value of each property

is first set to the median of all linked events and is then optimized by repeating the dynamic programming portion of the program until no further improvement of the value can be achieved. These procedures are then repeated for all rows of X.

4) Optimization of Finite State Machines: The above procedures create a finite state machine for each row of X. However, some clusters can be distributed to different loads simultaneously when the number of ones in any column of X exceeds one. These overlaps must be solved. Thus, the program examines overlaps for each column, and if overlaps exist, it selects the best row based on the quality value of the best path for each overlapping appliance. The algorithm then recreates the finite state machines for the non-best loads without the overlapped cluster. If the combinations of clusters cannot produce any valid sequences, then such loads are excluded from the matrix X. This procedure is repeated until all overlaps are solved and the algorithm has assembled the best-fit static table which can be returned to the meter. Once the table is returned, the learning phase is complete.

D. When to Learn

The schedule for learning consists of both reactive learning initiated from the meters and proactive learning initiated from the controller. When either detects the need for learning it informs the other and the meters begin to transmit data until the controller indicates that it has sufficient information to build a static table. If the meter finds that it has no static table (presumable on initiation), believes that a new unique load has been added, or detects an error in the static table then the meter will send a learning request. If the controller believes that the meter is in error, then it can issue a learning request. Loads no longer present in the static table should be removed in order to increase the monitoring efficiency on the meter. In general, it is difficult for the meter to detect that a load has been removed without conducting advanced analysis. Therefore if a threshold period is reached since the last learning phase, the meter will re-enter the learning phase in order to update the static table.

1) Bandwidth Constraint: Assuming the data is not encrypted, it should take a 24 byte payload per meter measurement. To send all the information back to the back end, it would require a bandwidth of 24 B/second and would need to send about 2.1 MB per day. While this does not seem excessive, there are thousands of meters reporting to each server, and this is not the only service the network will provide. For example, in a typical ZigBee network, payload bandwidth is limited to an optimal 120 kbps or 15 kB per second. Thus, if each meter has a 24 B payload, about 625 meters could concurrently talk to each ZigBee transmitter under ideal conditions. However, noise further reduces these rates. Therefore, reducing the bandwidth requirement is critical to making this a viable solution. One such way to reduce the bandwidth requirement will be finding an optimal learning frequency so as not to congest the network.

2) State Table Error Detection: An static table error detection module was implemented on the meter to ensure a high-fidelity operating environment. These errors can exist due to either newly added loads or newly seen existing loads which remained off during the previous learning phase. Three types of errors can be detected in the detection module:

(1) If the absolute difference between the sum of the power from the entries computed by the load monitoring module and the currently detected total power is greater than some threshold, some loads may be missing in the static state table. The meter currently uses a threshold, which is two times the sum of the standard deviation of all possible states of all loads.

(2) If a load changes state too often, it is a strong indication of an error in the static table; the acceptable change rate is load-specific. For example, it is reasonable to observe frequent changes in an air-conditioner or a heater, but it is unreasonable for a car battery charger.

(3) If more than a certain number of loads change states during one edge event, an error is indicated.

If any one of the three behaviors is observed, the error detection module will trigger its alert routine and send a request to the controller indicating a need for relearning the static table. We assume that the algorithm that builds the static table finds similar appliance profiles over multiple learning phases regardless of the amount of churn in the system. Based on this assumption, the controller examines the load tables created from multiple sets of data. If it finds a signature whose state transition profile is different from the previously detected loads, it judges that a new load has been added. The controller ends its learning period when it does not find new signatures for a specified period of time. The time and frequency of the learning can be adjusted to avoid congestion on the communication network. This process can be optimized if necessary; for instance, it may be sufficient to focus on periods of heavy use since these periods provide more data in less time. Furthermore, when requesting to enter the learning phase, the meters may assign reasoning (a label containing the type of error) and priority (urgent or not-so-urgent) for each detected error and embed these labels into the relearning request. This would facilitate the controller in coordinating the relearning phase of the entire system in order to avoid congestion by prioritizing important errors. For example, if the algorithm is looking for larger loads, it is reasonable to ignore errors caused by small load signatures.

The additional bandwidth due to this scheme is negligible when not in learning phase. The requests for relearning can be sent with the keep-alive heartbeats requiring modest bandwidth. Additionally, the bandwidth required to conduct the learning phase can be monitored and controlled by the backend controller since it instructs the meters when to start and stop sending the learning information. Thus, the scheme is both controllable and robust.

E. Building the Dynamic Table.

The meter has a load monitoring module to identify the current states of all loads in the static state table upon detecting a real-time edge event, and keeps the latest state information in a dynamic table. Analysis through either a knapsack or an

incremental algorithm are two commonly used techniques for appliance monitoring [16]. The knapsack algorithm searches for a combination of loads whose sum power is maximized under the constraint that the total power is less than the current observed power. The incremental analysis determines which load changed state based on the total power change observed in each edge event. Error propagation is one major drawback of the incremental analysis. Comparatively, running the knapsack algorithm continuously in real-time is computationally intensive. Therefore, we propose a hybrid design, which runs a modified knapsack algorithm on each edge event.

The appliance monitoring process can be formularized as a 0-1 knapsack problem:

$$Max : \sum_{i,j} w_{i,j}x_{i,j}, where \sum_{i,j} w_{i,j}x_{i,j} \leq W + T$$

where

W	real power observed in an edge event
T	tolerance value, the sum of the standard deviation of all possible loads given the current detected power
$x_{i,j}$	on/off of state j of appliance i, $x_{i,j} \in \{0, 1\}$
$w_{i,j}$	real power consumption of load i in state j

A brute force algorithm to detect the optimal combination given a total of n states across all loads in the static state table requires complexity $O(2^n)$. By comparison, the algorithm described above has complexity $O(n(W + T)/M)$, where $W + T$ is the total weight, and M is the minimum detection power unit. For example, if we have 10 loads with 2 states each (so $n = 20$) and the values $W = 5000W$, $T = 500W$, $M = 100W$, then the brute force method requires $O(2^{20})$, while our algorithm requires only $O(20 \times 55)$. This algorithm works well if three conditions are satisfied: (1) there is an accurate static table, (2) each load has discrete finite states, and (3) no two loads have the same power signature.

III. Algorithm Analysis

Our proposed dNLIM algorithm must ensure that the meter data can be compressed reasonably enough to send over limited bandwidth links, that the meters can identify loads based off of an accurate static table, and that a reasonable learning phase will allow the controller to build an accurate static table for the meters. Ideally, the meters, running with limited learning, will be able to identify the same loads as the fully functional NILM on the controller.

A data processing flow was created in order to test these requirements. The event detector and the real time monitor were implemented on an AVR32 simulator with the UC3A0512 chip and 512KB of memory. All the backend modules were implemented using Java on an Intel Xeon 2.13GHz desktop with 4GB of memory. Our approach was first tested using data taken in the lab using an SEL-734, which can report readings up to ten times a second for both real and reactive power.

After the initial testing phase, we then tested on data from a test home. The test home was monitoring its power usage using a TED 1000. This meter allowed us to take real power readings once a second and was useful in allowing us to test the full process from start to finish. Each portion was first tested individually; then the whole system was tested from beginning to end. The end to end tests were conducted by first running the learning phase to ascertain the major appliances and then comparing the output of the meter's monitoring algorithm against the more complex genetic algorithm. The results of the testing are summarized below.

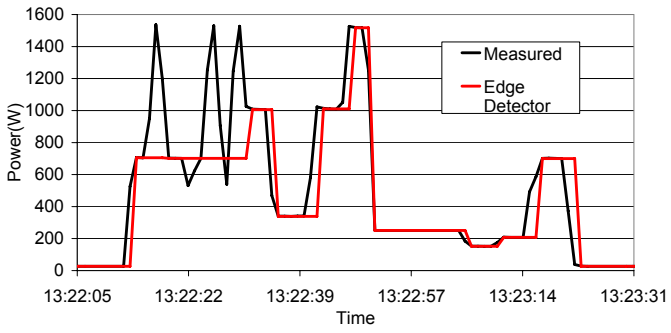

Fig. 2. Event Detection Algorithm

A. Event Detection

The event detection algorithm was tested on data collected both in the lab and on site. Output produced by running the event detector against lab data is shown in Figure 2.

As the figure shows, the event detector is able to ignore transient energy spikes. Power readings can be rather noisy and as such, transient points may introduce incorrect static and dynamic state tables. This particular figure has 101 data points for the measured power and 12 data points for the event detection, thereby achieving nearly 88% compression. In the household, where appliances switch states less frequently, we have achieved upwards of 99% compression.

B. Learning Phase Testing

1) Clustering: The biggest problem that faced the clustering algorithm was what magnitude of granularity would exist between separate edge events. If the distribution of appliance on/off events was uniformly distributed in the plane of real power, it would have been difficult for the clustering algorithm to make intelligent decisions based upon the data. Extensive testing revealed that appliances over 1000 Watts seemed to provide enough granularity to be detectable.

Running against data from the test home, the algorithm produced eight clusters, all separated by at least 250 Watts, a configurable threshold. It also recorded the corresponding number of on/off events associated with each. Judging from our results, events below 500 Watts, or so, are not easily distinguishable by the clustering algorithm. Thus, an appliance must consume at least this much power for it to be detectable under a scheme based solely on real power. This data is then passed on to the static table generation procedure.

2) Appliance Table Generation: The appliance table generation algorithm was tested using data from a controlled test

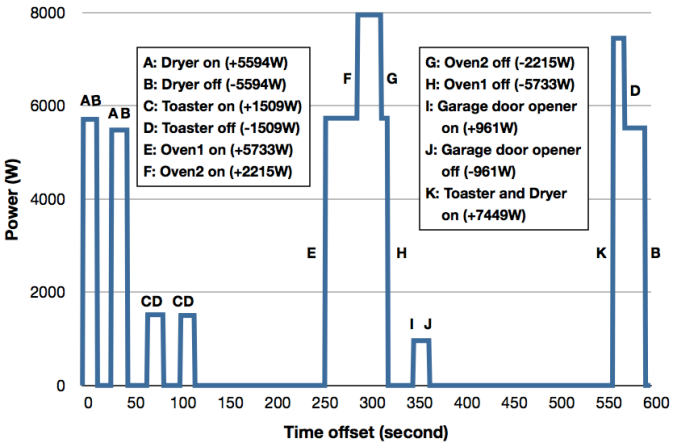

Fig. 3. Test Input Data

Appliance	#States	State 1 Real Power (W)		State 2 Real Power (W)	
		Mean	Std	Mean	Std
(A) Dryer	2	5594	200	2510	90
(B) Oven 1	1	5733	65	NA	NA
(C) Oven 2	1	2215	74	NA	NA
(D) Garage	1	961	55	NA	NA
(E) Toaster	1	1509	45	NA	NA

(a)

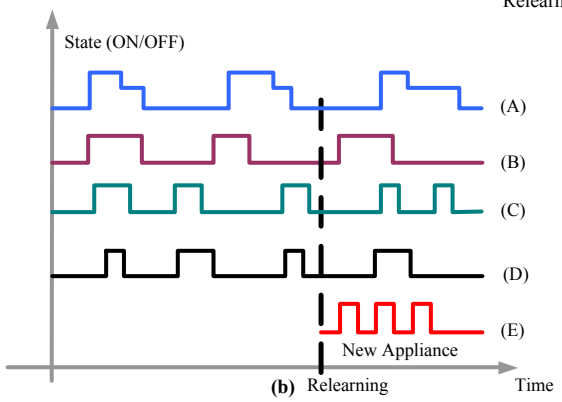

Fig. 4. (a) Static Table (b) Dynamic Table Over Time

conducted at the test home. The input edge events and corresponding clusters were generated as an ideal representation of the collected data. The input data is illustrated in Figure 3. From this input, the algorithm correctly identified the five appliances and created a corresponding static state table, as seen in Figure 4(a).

To analyze the algorithm, let the on/off transitions be represented by [on,off]. In this test, our program did not produce an appliance with state transitions [K, D+B] even though that pattern actually appeared in the input. The reason for this is because D and B belong to other appliances (i.e., a toaster and a dryer) whose transitions are [C, D] and [A, B] respectively. Since the patterns [A, B] and [C, D] appeared more frequently than the pattern [K, D+B], the appliance with transitions [K, D+B] was excluded by the optimization phase of the algorithm that solved the overlapping clusters. Another point worth mentioning is that the program produced two appliances, an oven 1 and an oven 2, whose transitions were [E, H] and [F, G], respectively, instead of producing one appliance whose transitions were [E+F, G+H]. This occured because the quality value of the latter appliance with four states was smaller than the quality values of the other appliances.

Then, using this static table, we keep track of the appliance states by updating the dynaminc table at each time step, as shown in Figure 4(b), which can the backend can then query.

C. dNILM Analysis

Due to the nature of our distributed architecture, much of the computation must take place on the meters. In order to justify that dNILM preserves sufficient accuracy, we compare our distributed scheme against a purely centralized scheme.

The distributed scheme we evaluate is the one we have discussed previously – i.e., the meter runs the appliance monitoring algorithm and has the ability to relearn once it detects a high error rate. We also discard the two lowest power appliance profiles that the backend generates. We do this because their values of 300W and 700W are below 1000W, what we generally consider to be noise. We then produce a single static state table from one day's worth of data and generate the dynamic state tables for the following days.

The centralized scheme utilizes purely the analysis offered by the finite state table generation algorithm at the backend. This is the scheme that requires higher computational power to run and thus cannot be run on the meter. We can treat this as the golden standard, as it affords a much higher accuracy as to which appliances are active at any given time.

Seen in Figure 5, we compute the accuracy as a/b, where a = distributed scheme accuracy and b = centralized scheme accuracy. We can see that on the first day, accuracy is at 77% when comparing the distributed scheme to the centralized scheme. The accuracy drops as the week goes on, bottoming out at 60% on the 26^{th}. On the 27^{th}, sufficient error rate is detected, and relearning occurs, allowing the meter to achieve 79% accuracy on the 28^{th} and peaking at 90% on the 31^{st}.

As for the running times of our algorithms, the monitoring algorithm is both short and linear with respect to the number of events. In fact, a day's worth of events requires on the order of 10 to 30 seconds to run on the meter and is therefore suited for use on the low-end meters. In contrast, if the data were to be transmitted to the backend for computation, this could put serious bandwidth consumption on a communications network that must also serve other purposes. Once on the backend, we will also run into the problem of scale as a single backend may have a service area of hundreds of thousands of meters. Even if computing a day's worth of data for one meter took one second, this would still require more than a day (almost 28 hours) to compute 100,000 clients. Another reason for using the meter for monitoring is that it can process information as

it reads it from the power line and thus always has an up-to-date view of the household. In contrast, the centralized solution could not answer queries in real-time.

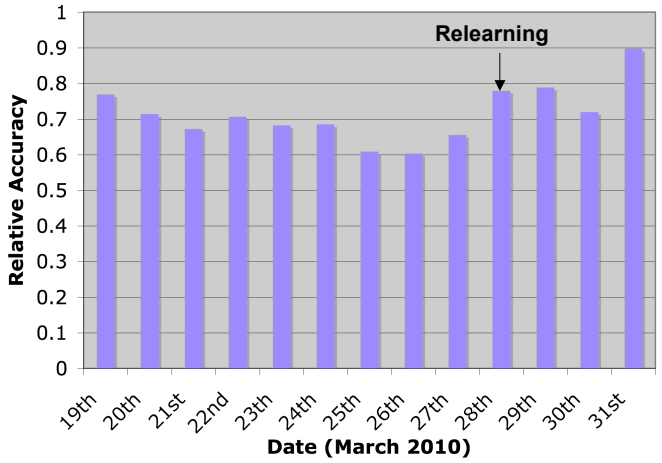

Fig. 5. Accuracy of dNILM compared to NILM

As for the learning algorithm, we note that its running time is supralinear, due to the combinatorics involved in generating the possible sequence patterns. Thus, it ought to be run on a computer that can handle its longer running time as well as provide the proper library support for genetic algorithms. Also, since it increases supralinearly in regard to the length of the learning period, this motivates both a need to investigate the optimal training length for generating an accurate static table and a need to investigate how to load-balance the relearning across a large network of meters.

D. Scalability Analyses

For dNILM to be viable, the methods proposed must be scalable both in bandwidth and processing power on AMI networks. Throughout experimentation, the test home yielded an average of 1,000 step events or roughly 15 kB of data per day. Assuming a highspeed backhaul network, bandwidth is primarily limited between the meters and backhaul points. With 4,000 meters per backhaul point, the bandwidth requirement would be roughly 4,000 $\times$ 15 kB = 60 MB of traffic per day or 5.5 kb/s of bandwidth. Assuming the use of a network such as ZigBee operating at a frequency of 2.4 GHz yielding 250 kb/s, the algorithm only consumes 2.2% of network bandwidth. Even assuming a slower ZigBee network with only 30 kb/s of bandwidth and a three-hop deep mesh network yielding an effective bandwidth of 10 kb/s, the algorithm can still support re-learning for every node every day using 55% of the total network bandwidth. Of course, these are worst case estimates since learning allows the bandwidth requirements to be reduced. Using the 10 kb/s ZigBee network with each node learning one day a week, the algorithm would require 7.9% of network bandwidth and re-learning one day a month would require 1.8% of network bandwidth.

Regarding the computational cost, the primary bottleneck is during the learning phase in the ESP's control center. Although it depends on parameters, we needed less than a minute in our tests to build a static state table from one day of data on a typical desktop PC. This implies that about 24 $\times$ 60 = 1,440 homes can be handled per day with one machine. Thus, assuming the backend carefully schedules learning in order to fully utilize computational resources, and the same 4,000 meters per backhaul, each compute cluster would only require 4,000 / 1,440 = 3 cores. With desktop parts supporting up to six cores on a single die and server parts supporting even more, this is hardly a substantial obstacle to overcome.

IV. Related Work

A significant amount of research has been conducted in the area of NILM over the past 30 years beginning with the initial concept and simple learning technique introduced by Hart et al. [1] Since then, much work has been done to improve NILM primarily through load identification using signatures of transient behavior through spectral frequency analysis. These methods are computationally similar to the performance of the adaptive learning technique in that they require more computational power than is provided on the current smart meters. Spectral analysis techniques assume a meter capable of taking measurements at a minimum of 60 Hz with an ideal capability of 120 Hz and 240 Hz optimal in order to analyze the transient phase of a device powering on or off. This is generally a unique signature that allows great accuracy in the measurement and highly accurate identification of loads. [2], [5]–[10], [12], [13]

Unfortunately, the architecture of the smart grid severely limits the usefulness of the recent NILM techniques. First, smart meters lack the available computational power to conduct the spectral analysis locally at the home. Bandwidth limitations are an important consideration in any design. While scalability is a primary concern when sending one meter reading per second, transient analysis requires between 60 to 240 times more data and bandwidth. This limitation has been identified and work has been done to introduce novel ways to compress the data. [11] While this may address the bandwidth problem, it still assumes a meter that can take high frequency measurements. Unfortunately such meters are more expensive than the smart meters being currently deployed. Thus, to implement such techniques, vendors would need to be convinced to deploy more expensive meters capable of high frequency measurements.

The goal of our work is to solve the bandwidth problem, the computation problem and the limited meter problem in one algorithm. We believe this technique fills a hole in NILM research due the uniqueness of the smart grid architecture. Our work is based directly upon Harts contribution [1] due to the use of only step changes in real power. Our contribution is to introduce and improve upon recent adaptive learning and dynamic programming techniques [3], [4] in order to create an algorithm capable of running on limited meters with minimal bandwidth while still maintaining a high degree of accuracy.

V. Conclusion

Smart grid technologies will enable the information flow and efficiency required to enter a brave new world. However, the architectures required to harness these technologies are not yet in place. Here, we have described one such architecture that we believe will support many of the current visions for the smart grid. We have adapted simple event detection and appliance monitoring schemes for use on the meter and more complicated NILM techniques for use on a central system. We have then shown that, through decoupling the learning and the monitoring, we can achieve nearly the same accuracy at a large gain in processing power and little overhead due to data transfer.

Acknowledgment

This work was supported in part by DOE DE-0000097, NSF CNS 07-16421, NSF CNS 05-24695, and Lockheed Martin. The views expressed are those of the authors only.

References

[1] G. W. Hart, "Nonintrusive appliance load monitoring," *Proceedings of the IEEE*, vol. 80, no. 12, pp. 1870–1891, 1992.

[2] M. L. Marceau and R. Zmeureanu, "Nonintrusive load disaggregation computer program to estimate the energy consumption of major end uses in residential buildings," *Energy Conversion and Management*, vol. 41, no. 13, pp. 1389–1403, September 2000.

[3] M. Baranski and J. Voss, "Genetic algorithm for pattern detection in NIALM systems," in *Systems, Man and Cybernetics, 2004 IEEE International Conference on*, vol. 4, 2004, pp. 3462–3468 vol.4.

[4] ——, "Detecting patterns of appliances from total load data using a dynamic programming approach," in *Data Mining, 2004. ICDM '04. Fourth IEEE International Conference on*, 2004, pp. 327–330.

[5] K. H. Ting, M. Lucente, G. S. K. Fung, W. K. Lee, and S. Y. R. Hui, "A Taxonomy of Load Signatures for Single-Phase Electric Appliances," in *IEEE PESC (Power Electronics Specialist Conference)*, June 2005, pp. 12–18.

[6] K. D. Lee, S. B. Leeb, L. K. Norford, P. R. Armstrong, J. Holloway, and S. R. Shaw, "Estimation of Variable-Speed-Drive Power Consumption From Harmonic Content," *Energy Conversion, IEEE Transactions on*, vol. 20, no. 3, pp. 566–574, 2005.

[7] C. Laughman, K. Lee, R. Cox, S. Shaw, S. Leeb, L. Norford, and P. Armstrong, "Power signature analysis," *Power and Energy Magazine, IEEE*, vol. 1, no. 2, pp. 56–63, 2003.

[8] S. Leeb, U. Khan, and S. Shaw, "Multiprocessing transient event detector for use in a nonintrusive electrical load monitoring system," Feb. 1998.

[9] G. R. Mitchell, R. W. Cox, J. Paris, and S. B. Leeb, "Shipboard Fluid System Diagnostic Indicators Using Non-Intrusive Load Monitoring," *Naval Engineers Journal*, vol. 119, no. 2, pp. 109–119, Oct. 2007.

[10] E. Proper, R. Cox, S. Leeb, K. Douglas, and J, "Field demonstration of a real-time non-intrusive monitoring system for condition-based maintenance," *Electric Ship Design*, 2009.

[11] Z. Remscrim, J. Paris, S. B. Leeb, S. R. Shaw, S. Neuman, C. Schantz, S. Muller, and S. Page, "FPGA-based spectral envelope preprocessor for power monitoring and control," in *Applied Power Electronics Conference and Exposition (APEC), 2010 Twenty-Fifth Annual IEEE*, 2010, pp. 2194–2201.

[12] S. R. Shaw, C. B. Abler, R. F. Lepard, D. Luo, S. B. Leeb, and L. K. Norford, "Instrumentation for High Performance Nonintrusive Electrical Load Monitoring," *Journal of Solar Energy Engineering*, vol. 120, no. 3, pp. 224–229, 1998.

[13] S. R. Shaw, S. B. Leeb, L. K. Norford, and R. W. Cox, "Nonintrusive Load Monitoring and Diagnostics in Power Systems," *Instrumentation and Measurement, IEEE Transactions on*, vol. 57, no. 7, pp. 1445–1454, Jul. 2008.

[14] D. C. Bergman, D. Jin, J. P. Juen, N. Tanaka, C. A. Gunter, and A. K. Wright, "Non-Intrusive Load Shed Verification," *IEEE Pervasive Computing*, 2010, To appear.

[15] "Java Genetic Algorithm Package (JGAP), version 3.4.4," http://jgap.sourceforge.net/, 2009.

[16] M. LeMay, J. J. Haas, and C. A. Gunter, "Collaborative Recommender Systems for Building Automation," in *System Sciences, 2009. HICSS '09. 42nd Hawaii International Conference on*, 2009, pp. 1–10.