

Energy Applications for an Energy Box

John S. Donnal, James Paris, and Steven B. Leeb, *Fellow, IEEE*

Abstract—Changes in the electric utility will necessitate new needs and opportunities for monitoring and controlling electric power consumption and generation. Technical solutions exploiting these opportunities and answering these needs would ideally preserve best practices like reliability, privacy, efficiency, and flexibility. A nonintrusive load monitor (NILM) can serve as an ideal platform for constructing an “energy box” capable of sophisticated monitoring and control. This paper introduces a data processing and analysis framework, NILM manager. NILM manager creates a business model for handling power data by minimizing network bandwidth and placing intelligence and feature expansion in easily transmitted “energy apps.”

Index Terms—Data visualization, digital signal processing, distributed computing, distributed information systems, virtual private networks.

I. INTRODUCTION

ELECTRIC utility is a disarmingly simple phrase for one of the mankind’s greatest engineering achievements. In the heady rush for “disruptive transformations” and new business opportunities, we are perhaps especially well-served to consider the features and characteristics that underly the pivotal importance of the grid. Proposed changes should be inspected closely with an eye on maximizing and preserving: privacy; flexibility and “future proofing” in permitting introduction of beneficial hardware; compatibility with conventional information infrastructure; and the availability of actionable information for conservation, resolving disputes, and ameliorating pathologies in support of grid operation. The utility should continue to seamlessly delight the customer by underpinning quality of life. The utility is here to serve its customers, not vice versa. It may well be that best-modes for a future smart grid avoid a requirement for an “Internet of Things” for utility operation. Poor hardware architectures may encourage the introduction of poor software. Poor software with wide reach could compromise load and load schedule diversity and hasten the introduction of opportunities for unplanned, correlated operation, or events that have plagued other markets.

As a bellwether of enthusiasm, note that tens of millions of “smart meters” have been installed in the U.S. alone [1]. Provocatively for consideration, one might argue

that these meters are severely limiting in many respects. They are relatively expensive compared to traditional meters. They typically compute sample data at rates of at most a few times per second, and produce data streams that are coarse for diagnostics and load identification. Yet, these data rates are sufficient to create a substantial burden on information or network bandwidth infrastructure when transmitted. They provide little in the way of “smart” analysis or data reduction, and create new burdens on distal processing sites for rendering the data actionable.

The emerging vision of a “smart grid” relies on active distribution networks with new levels of both monitoring and control. Many experiments and proposals exploit an energy controller or “energy box” to serve as a gateway for monitoring and control of loads and distributed generation assets in homes or buildings (see [2], [3]). These experiments are often conducted with an eye on new market opportunities for aggregators who would presumably assist in the control and operation of the utility [4]. Extensive modeling and optimization efforts are being conducted (see [5] and its associated references) to demonstrate potential techniques and benefits for implementing active monitoring and control.

This paper proposes and demonstrates an energy box built around a high performance, low cost computer that remains as a monitor at a site, a nonintrusive load monitor (NILM) [6]–[8]. Nonintrusive meters can measure power consumption and harmonic information by sampling at rates well over 1 kHz. A NILM can identify exactly what loads are consuming power and provide the end user with an itemized summary of power usage [9]–[12]. In this paper, a NILM serves not only as a sophisticated monitor but also as a flexible controller informed by the nonintrusive monitoring algorithms. A custom high-speed time-series database, NilMDB, organizes data for efficient retrieval, and a powerful visualization and programming interface, NILM manager, permits secure reconfiguration and programming from anywhere in the world. NILM manager supports scripts or “apps” that can take advantage of sophisticated math libraries stored with the NILM. With this architecture, remote nonintrusive meters do not need a dedicated work station for analysis. Traditional smart meters require significant, dedicated bandwidth to transmit raw data back to the utility’s servers. On a NILM, raw data belongs to the facility owner and remains on the facility computer. This distributes the computation and in effect parameterizes, i.e., compresses the raw data into information. Using NILM manager, users can visualize power consumption, generate custom reports using an integrated scripting engine, and control loads all while using minimal bandwidth and without exposing their data to a third party. New capabilities can be installed on

Manuscript received October 5, 2015; revised January 26, 2016; accepted April 24, 2016. Date of publication April 28, 2016; date of current version September 8, 2016. This work was supported in part by the MIT Energy Initiative, in part by the Grainger Foundation, and in part by the U.S. Navy.

J. S. Donnal and S. B. Leeb are with the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139 USA (e-mail: jdonnal@mit.edu).

J. Paris was with the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139 USA. He is now with Bolt Innovation Management, Boston, MA 02111 USA.

Digital Object Identifier 10.1109/JIOT.2016.2560123

the box remotely with short, powerful scripts that update the function and utility of the NILM. The box exploits noncontact sensors that can be easily installed by unspecialized hands, and the entire facility monitor runs on a low-cost computer like a Raspberry Pi demonstrations of this system and information architecture for flexible and reconfigurable monitoring and control are presented.

II. SYSTEM ARCHITECTURE

Recording current and voltage with enough resolution to identify load characteristics requires sampling at relatively high rates. NILMs capable of interesting diagnostics and load recognition typically generate very large data sets. Equation (1) can be used to estimate the storage requirements for a typical installation

$$R = 2N_\phi \times f_s \times B_{\text{adc}} \quad (1)$$

where R is the data rate in bytes per second, N_ϕ is the number of phases (usually two for residential and three for industrial environments), f_s is the sampling frequency, and B_{adc} is the ADC resolution, or the number of bytes used to represent a sample measurement (usually about two). The product of these factors is multiplied by two because both current and voltage waveforms are recorded for each phase.

Using (1), a NILM running at $f_s = 8$ kHz will produce over 5 GB of data per day for a standard home. Data sets of this size are difficult to transmit over a residential network. In NILMs deployed in [13], for example, equipment operators mailed hard drives and DVDs back to the laboratory for analysis. The cost in resources and man-hours make this type of installation impractical for all but the most limited deployment scenarios. Even if data can be reliably collected, plotting the current and voltage over a single day involves billions of individual samples which is beyond the capability of many standard software packages (such as Excel). Previous work has focused on using signature detection to reduce the dataset size, storing only equipment “on” and “off” events instead of current and voltage [10]–[12]. However, such an aggressive data reduction step without a clearly defined outcome or monitoring objective artificially limits the utility of the NILM.

NILM manager, a cloud platform that enables quick and easy access to NILM data, solves the access and analysis challenges created by high bandwidth or “big data” power monitoring. A “remote” NILM is installed at a facility to be monitored. Desktop-power computing is readily available in “deck of cards” sized hardware that can be installed quickly at a site with terabytes of local storage, at prices comparable to those of a modern solid state electricity meter. Data collected by this remote NILM is never fully transmitted from the site, minimizing network traffic. Rather, data are managed locally on the site computer by custom high-speed database software, NilmDB, described in [14]. NILM manager provides a central management node that connects multiple remote NILMs with a virtual private network (VPN) and hosts a website that allows authorized users to view and analyze data collected by NILM systems.

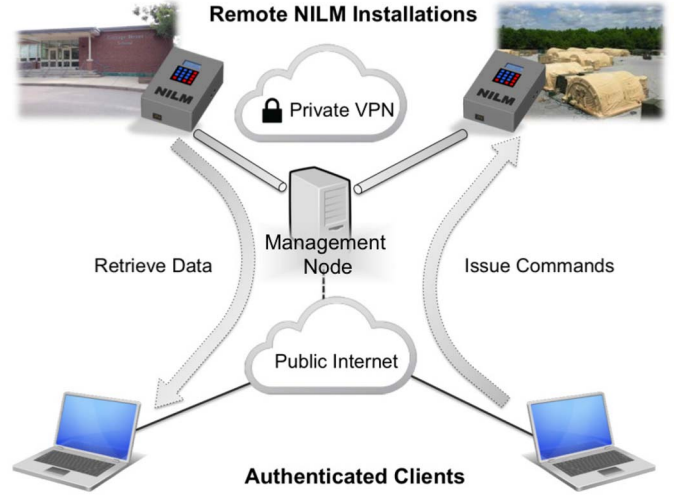


Fig. 1. NILM manager system architecture. The management node relays requests from authenticated clients to remote NILMs over a secure VPN. Clients can issue commands and retrieve data over a Web interface.

A. NILM Virtual Private Network

NILM manager controls the computing “center” of a virtual private network that securely connects remote NILMs, each running NilmDB, to the management node. The network is virtual in the sense that all communication occurs over the public Internet but is encrypted so only NILMs and the management node can decipher the content. Extensive computation on acquired data are relegated to local computing managed by NilmDB at a site. New programs or energy apps can be downloaded from NILM manager to a NilmDB installation. New analysis results can be uploaded from a remote site to NILM manager for Web presentation, which can of course be through secure connections. Small energy apps and small reports or analysis results, typically a few kilobytes, can provide full, powerful access to remote high bandwidth data with minimal network data requirements. A (low technology) cell phone can and has provided more than enough bandwidth for managing a full industrial monitor in our experiments.

Fig. 1 shows a conceptual view of the NILM VPN. Users can request data from a NILM and send it commands all without any physical access to the machine. The management node coordinates VPN traffic and ensures that only authorized users have access to NILM systems.

B. Web Platform

Users interact with NILMs through a website hosted by the management node. The website is available over the public Internet which means it is accessible from any connected device including tablets and cell phones. Users authenticate with a username and password although certificate based authentication or other forms of protection could be implemented if additional security is required.

By presenting users with a Web interface rather than a direct connection (e.g., via SSH) to the remote meter, the user interaction tools (NILM manager) are decoupled from NILM system tools. This means NilmDB and other backend software

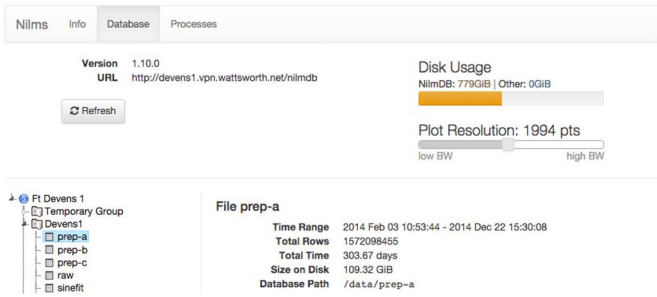


Fig. 2. NILM configuration interface. The plot resolution slider controls how many data points are displayed in data visualization tool (shown in Fig. 3).

on the NILM can be updated without affecting how the user interacts with the NILM data.

III. DATA VISUALIZATION

One of the primary difficulties in nonintrusive load monitoring is visualizing the high bandwidth data collected by the current and voltage sensors. A NILM produces thousands of data points each second. Tools such as Excel and MATLAB consume significant system resources to produce plots for datasets of this size. Complicating matters further, NILMs often have limited network bandwidth making transmission of the raw data to a workstation difficult or impossible. NILM manager solves this problem by using a decimation algorithm to visualize large datasets.

A. Stream Configuration

NILMs connected to the management node are configured through the Web interface shown in Fig. 2. This interface presents the data collected by the NILM as a series of files organized into directories. Users can navigate through the data on the NILM just as they would navigate folders on their desktop. While they appear as flat datatypes on the Web interface, each file corresponds to a hierarchy of streams on the NILM itself.

As the NILM adds data to a stream, it simultaneously computes a decimated child stream. For every four elements in the parent stream, the child contains a single [min, max, mean] tuple. This process is performed recursively with each successive child containing a factor of four fewer elements than its parent. When this process is carried out to completion (the final child containing only one sample), the total storage requirement only increases by a factor of two [14].

The plot resolution slider in the top right of the interface sets the number of data points returned by the NILM when a user requests an interval of data. NILM manager checks how many data points are contained in the requested interval and returns the lowest decimated child stream that fits within the configured plot resolution. The raw data is only returned if the interval requested is small enough that there are fewer raw samples than the plot resolution setting.

B. Presentation

The NILM manager website provides an intuitive plotting interface shown in Fig. 3. Note that this figure shows real

data from a monitoring site in our research program. The system has been running for nearly two years, and the remote monitor contains tremendously detailed data, down to the envelopes of individual electrical transients as shown in the figure. Nevertheless, access to this data are almost instantaneous from any Web connection anywhere. The interface uses decimated streams to allow users to view any dataset from any remote NILM at any time scale. Panning and zooming through the data operates like Google Maps with progressively higher resolution data returned as a user “zooms in” to a particular area of a waveform. Progressive views are delivered essentially instantaneously.

The plotting interface is implemented in Javascript which runs in the client browser. The code is based on the open source “Flot jQuery” plugin although it has been highly customized for this application [15]. The plotting code has three display modes. If the time interval is short enough that the raw data fits within the plot resolution setting, a simple line graph is displayed. If, as is usually the case, the raw data contains too many samples, data from the selected decimation level is displayed as a [min, max] envelope around the mean which is plotted as a line graph. The envelope is the same color as the mean with added transparency. This provides feedback about the structure of the data without obscuring other time series on the same plot (as in the case of MATLAB or Excel). Finally, if a time interval contains too much data in all available decimation levels (which occurs when a NILM has not yet decimated a new stream), a thick horizontal line is drawn in place of the data and an asterisk is added to the legend indicating the inability to plot the particular stream at the selected time scale. Fig. 4 shows the three plot display styles. The client code automatically switches between styles as the user navigates between datasets and timescales.

IV. DATA PROCESSING

Current smart meters typically transmit their measurements wirelessly to a central monitoring node which limits their resolution, as these links generally cannot carry sufficiently large amounts of data [16]–[18]. Exposing raw data also exacerbates privacy concerns. The on-board CPU cores in even a low-cost NILM process data locally. Data need not be moved in bulk from the monitoring site. Short, actionable reports and analyses can be transmitted to a facilities manager or service provider as privacy restrictions permit. The information can also be used locally for control. Moving computation from a centralized server to a distributed embedded environment requires an efficient data processing framework. The following sections describe this framework, and illustrate how apps on distributed NILM energy boxes can analyze, report on, and control power systems.

A. Management and Preprocessor

NILMs support remote management through a specialized application programming interface (API), which allows clients to upload and execute custom scripts. This API is exposed to the management node over HTTP with security provided by

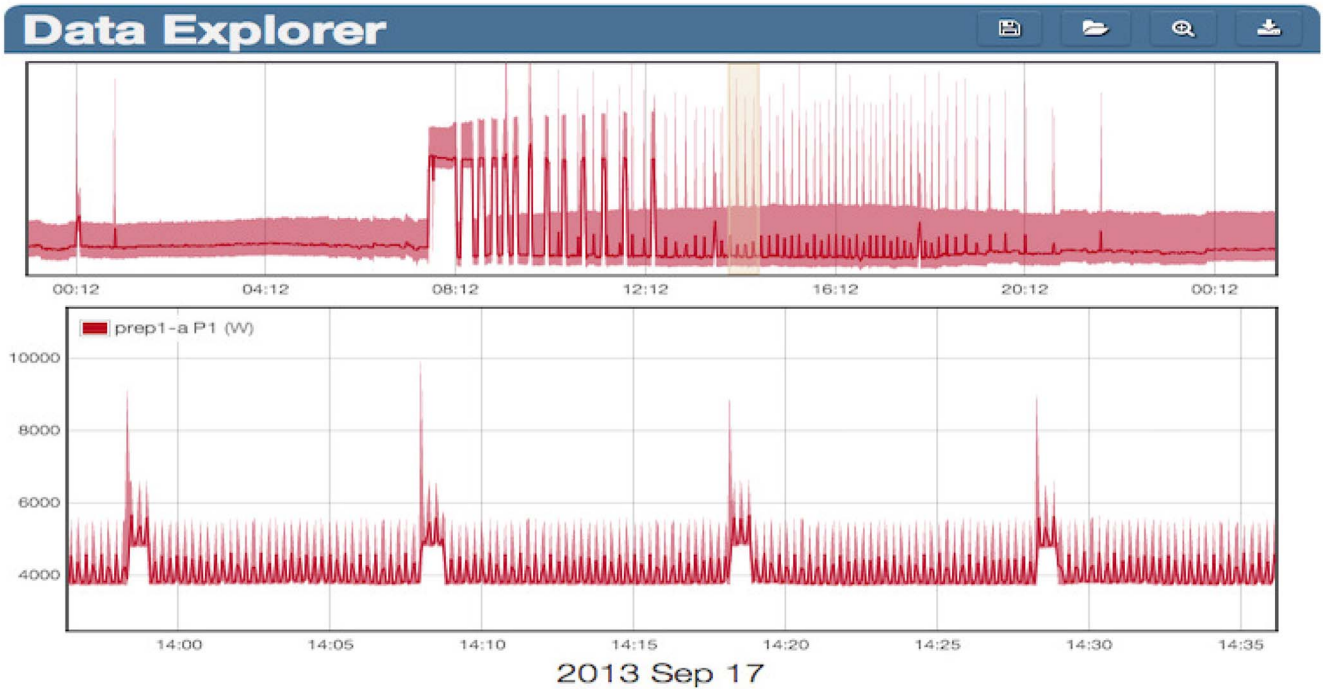


Fig. 3. Data visualization using the Web plotting tool. Top: 24 h of power data. Bottom: higher resolution view of the highlighted segment. This view represents 5.1 M samples but is drawn using just 2 K decimated samples (less than 0.05% of the raw samples).

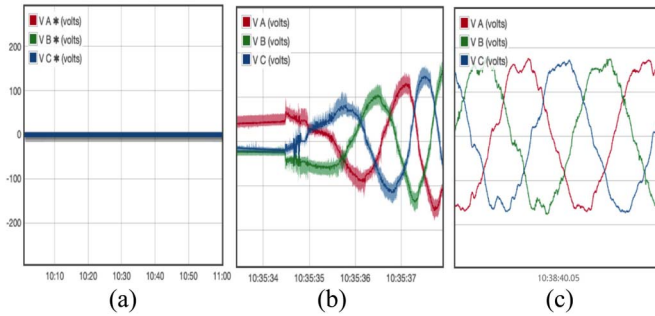


Fig. 4. NILM manager automatically adjusts the plot type based on how many points are in the selected dataset. (a) When a stream has too much data and no available decimations, a solid line indicates the plot cannot be displayed. (b) If decimations are available an envelope of the dataset is shown, and (c) if the time interval is short enough, the raw data is plotted directly.

the VPN tunnel. The management node uses this API for system administration tasks such as database cleanup, software updates, and system diagnostics. The management node establishes a sandbox on top of this API in which end users can execute their own scripts called energy apps. These scripts use input hooks to link to data streams stored on the NILM. An app can use data from multiple streams each of which may have different intervals of data and sampling rates. The NILM runs a two stage preprocessor that consolidates input data from diverse source streams into a single time stamped array which makes it easier to write energy processing algorithms.

1) *Multistream Wrapper*: Data streams may be electrical measurements, data from secondary sensors, or outputs from other NILM processes. For processes that require inputs from multiple streams care must be taken to schedule the process

appropriately and only run it over time intervals where all of its input streams are available. Sensor data may arrive in bursts with significant lag, and streams produced by other processes create scheduling dependencies. The multistream wrapper manages these dependencies and ensures that a process is only run over intervals where its inputs are available.

2) *Resampler*: Once the input streams have been assembled, the resampler produces a single composite data set with timestamped rows where each column is a process input. If all inputs come from a common source stream then this array is straightforward to assemble, however, apps using inputs from different streams generally require resampling. For example, an app that uses outside temperature and real power consumption as inputs (to compute energy usage as a function of weather for example) must use either down-sampled energy measurements or up-sampled temperature measurements. When multiple streams are used as inputs the user specifies a “master” stream and resampler runs a linear interpolator or a decimator on the other inputs to create a uniformly sampled dataset.

The stream iterator framework makes it easy to write custom applications that use this dataset to run analysis and control algorithms.

B. Stream Iterators

Each energy app is based around a stream iterator which enables computation on large NILM datasets. Traditional iterators such as `for` and `while` loops operate on static datasets, but NILM data arrives continuously. Stream iterators provide the ability to operate on continuous datasets by combining a traditional looping iterator with a persistent state. When the

```

1 a = [...]; b = [...] #filter coeffs
2 def setup(state):
3     zi = [...] #initial state for filter
4     state.initializeSlot("filter_zi", zi)

```

Listing 1. Setup function for a NILM stream iterator.

```

1 #data is 2 column array: [timestamp, sample]
2 def run(data, state, insert):
3     #initialize filter with saved zi values
4     zi = state.retrieveSlot("filter_zi")
5     #run filter against this chunk of data
6     (y,zf)=scipy.signal.lfilter(b,a,data[:,1],zi=zi)
7     data[:,1]=y #update data in place
8     insert(data) #save output
9     state.saveSlot(zi,"filter_zi") #update zi

```

Listing 2. Run function for a NILM stream iterator.

stream iterator has finished processing the available data it saves its state variables so that when it runs on the next chunk of data, it can pick up exactly where it left off. This allows the programmer to treat the datasets as continuous streams while giving the NILM flexibility to choose chunk size and processing rate based on the available system resources.

Building a stream iterator is a two step process. First, the user defines a `setup` function (see Listing 1). This function initializes a state object which provides persistent storage between process runs. Data are stored in slots which are accessed by string identifiers, similar to a dictionary. This function only runs the first time the app is executed. The setup function in Listing 1 initializes `state` for an example app which runs a linear filter on a NILM data stream. The filter coefficients do not need to be stored in `state` because they are constants which do not change between runs of the process.

After initializing the app state in `setup`, the user then defines a `run` function. This function receives the resampled input streams from the preprocessor and performs the actual data processing (see Listing 2). In this function traditional iterators and third party libraries can be used to build complex signal processing algorithms. Listing 2 shows a simple example which runs a linear filter using SciPy, an open source Python library. More advanced code could perform load identification, equipment diagnostics, or a variety of other data analysis. The `insert` argument is a function handle for saving results to an output data stream. After processing the input, any variables that should persist between runs are stored in the state object. The NILM repeatedly runs this function as more input data becomes available.

C. Reports

In addition to generating output data streams (such as the filter example in Listings 1 and 2), energy apps can also produce reports. Reports run over a specific interval of data and produce an HTML document that can contain custom text, plots, and tables. After the stream iterator has processed the specified duration of data (e.g., hour, day, week, etc.), an HTML generator produces the report document. A report is defined by an analysis function and an HTML template. The analysis

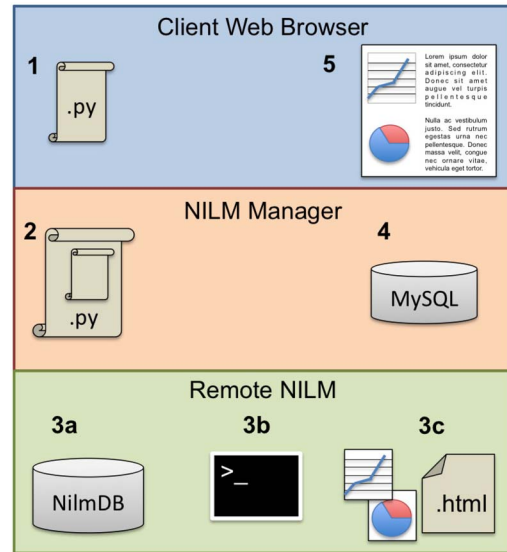


Fig. 5. Stages of a NILM report. 1: The end user designs a report in the Web IDE. 2: The management node adds support code to build an executable script which it then sends to the remote NILM. 3a–b: The script links to streams in the NilmDB, and runs to completion. 3c: The HTML report and associated figures are sent back to management node. 4: The management node stores the report in its MySQL database. 5: Authorized users can view the report in their Web browser.

function uses the process `state` to compute summary statistics and figures. These are injected into the report template to create a full HTML document.

Fig. 5 shows the process of creating an energy app report. In step 1, the user defines the stream iterator, analysis function, and HTML template. Next, the management node adds the support code making an executable script which is sent to the target NILM. In steps 3a–3c the NILM runs the energy app which generates an HTML document and associated figures. The NILM returns these files to the management which stores them in a MySQL database and makes them accessible through the Web interface to authorized users. Hosting the report document on the management node rather than the NILM insulates the NILM from external network traffic providing an additional layer of security and reducing the demand for its limited bandwidth. If privacy is a greater concern than network bandwidth, the NILM can retain the report in local storage instead.

D. NILM Manager IDE

The NILM manager website provides a complete integrated development environment (IDE) to write, test, and deploy energy apps. Fig. 6 shows the app designer interface. The left hand panel is a syntax highlighting code editor with multiple tabs for app initialization, stream iterator definitions, and report templates. To run the app in development mode the user selects input streams and a time range using the plotting window on the bottom right. Text output generated by the app is continuously retrieved from the NILM and displayed in the upper right hand panel. This panel also displays debugging information in the event of an error. In development mode the

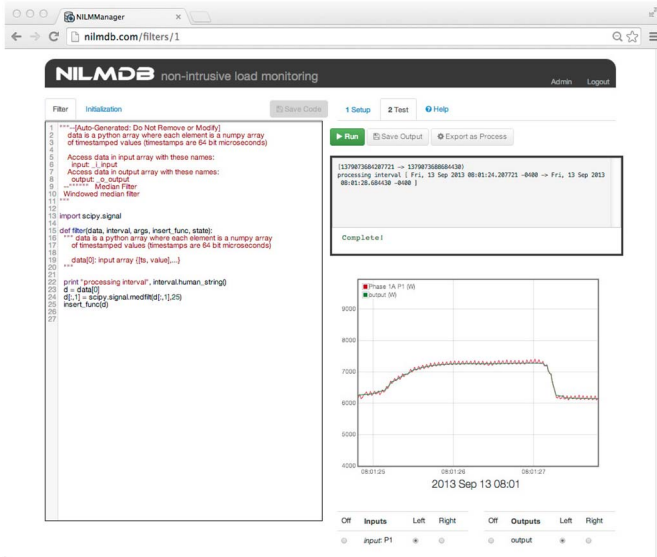


Fig. 6. NILM manager IDE for designing energy apps.

output stream is temporarily allocated on the remote NILM, and each time the app runs, it overwrites the previous output.

Once the user is satisfied with the app's performance, the code can be deployed to one or more target NILMs and scheduled as a continuous process. The management node tracks deployed processes and archives system logs and metrics so that users can manage the computational resources on their NILMs appropriately. When an app is deployed in production mode its output stream is permanently allocated on the target NILM and made available to other users either to plot or use as an input to other apps.

V. DESIGNING ENERGY APPS

Users with appropriate security permissions can now design useful applications on the NILM to monitor and control their power systems. Energy apps run entirely on the NILM itself and do not rely on external services or high bandwidth network connections. The following example shows examples of how these apps are designed at real monitoring sites.

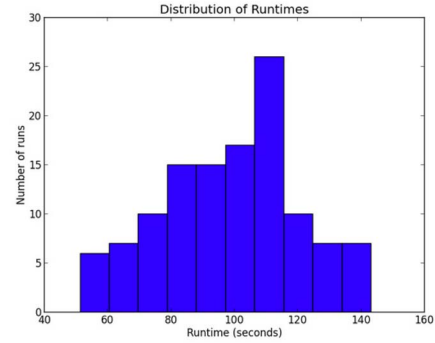
A. Cycling System App

Reports present actionable information to end users turning NILMs into powerful monitoring and diagnostic tools. Consider a standard cycling system such as a shop air compressor. This system requires periodic maintenance based on hours of operation and excessive runtimes may indicate leaks or abnormal usage, but adding sensors to track air compressor runs is generally too expensive for the benefit it provides. NILM is the cost effective solution. A single NILM can monitor multiple air compressors, and indeed any electric machine in a shop, eliminating costly (and maintenance-prone) sensor networks [19].

Fig. 7 shows an example of a report for tracking trends in air compressor runtime. The report is built in two stages. First a stream iterator, defined by `setup` and `run` functions,

Report for Air Compressor

120 runs for a total operating time of 197 minutes



Machine status: maintenance required

Fig. 7. Example of a NILM report for monitoring an air compressor (generated by Listing 4).

```
def analyze(state, save_fig):
    #retrieve data calculated by [run] function
    runtimes = state.retrieveSlot("runtimes")
    #calculate statistics
    mins = int(np.sum(runtimes)/60) #minutes
    state.initializeSlot("time",mins)
    state.initializeSlot("runs",len(runtimes))
    #if any runtime > 3 hours raise alarm
    if(np.max(runtimes)>180):
        state.initializeSlot("status",
                            "maintenance required")
    else:
        state.initializeSlot("status","OK")
    #make a histogram of the runtimes
    fig = plt.plot(runtimes)
    save_fig("runtime_histogram",fig)
    #...additional plot formatting not shown
```

Listing 3. Analyze function for a report process.

processes data over a specified time interval. The stream iterator identifies machine turn on and turn off events by tracking transients in the power waveform. When a machinery run is detected it is added to an array stored in the process state.

After the stream iterator processes the data, the analysis function in Listing 3 generates summary statistics and builds a histogram of machine runtimes. The statistics are added to the process state, and the `saveFigure` function saves the plot using a similar string-tag syntax.

Finally, the HTML generator builds the report using the template shown in Listing 4. Markdown is used for simplicity although raw HTML and CSS can be mixed in for finer grained control of the document format. Content from the process state is injected into the template using double braces `{{...}}`, and the `insertFigure` command embeds plots as HTML images.

The HTML document and plot image are sent back to the management node and hosted through the Web interface. Reports like this example, can be scheduled to run once or run continuously. When set for continuous operation the user specifies a repeat interval and duration. For example a report can be set to run every hour using the past 24 h of data. The Web

```

1 Report for Air Compressor
2 -----
3 {{runs}} runs for a total
4 operating time of **{{time}}** minutes
5
6 {{insertFigure("hist")}}
7
8 #####Machine status: {{status}}

```

Listing 4. Template for report HTML.

Identify transients:

Off	Exemplars	Left	Right		
<input type="radio"/>	Laser On	<input type="radio"/>	<input checked="" type="radio"/>	<input type="button" value="Edit"/>	<input type="button" value="Remove"/>
<input type="radio"/>	Laser Off	<input type="radio"/>	<input checked="" type="radio"/>	<input type="button" value="Edit"/>	<input type="button" value="Remove"/>
<input type="radio"/>	Compressor On	<input type="radio"/>	<input type="radio"/>	<input type="button" value="Remove"/>	<input type="button" value="Remove"/>
<input type="radio"/>	Compressor Off	<input type="radio"/>	<input type="radio"/>	<input type="button" value="Remove"/>	<input type="button" value="Remove"/>

Fig. 8. Training the load identifier on shop equipment. The cross correlator presented in [14] uses exemplars to identify turn-on/off events of machines. The exemplar for “Compressor Off” is shown in the popup window.

interface provides a navigation tool to browse series of reports which can be help identify trends and spot abnormalities in equipment operation.

B. Power Quality App

In modern machine shops sensitive devices like CNC tools and 3-D printers are collocated with other large equipment that can interfere with the line voltage causing droops and harmonics. In this experiment, a 3-D printer shares shop space with a laser cutter and an air compressor, both of which introduce power quality problems including voltage sags. Shop preference is to avoid sharp voltage sags of more than two volts during operation of the 3-D printer. A NILM monitors the aggregate current and voltage for the entire shop. Fig. 8 shows the power consumption of the shop during normal operation. A cross correlator (discussed in [14]) is trained to identify these loads. The turn-on and turn-off events are indicated in Fig. 9 by colored bars. Here, four transients are identified corresponding to a run of the laser cutter and air compressor, respectively. The lower power cycling waveform is the PWM bed heater of the 3-D printer. Energy apps on the NILM can both quantify the shop’s power quality and improve the power quality to the 3-D printer during operation by ensuring proper scheduling of the loads.

Over this time interval the NILM detected voltage disturbances large enough to interfere with the 3-D printer’s operation. Fig. 10 shows the power waveform as well as the line voltage as measured by the NILM. The app outlined in Listing 5 identifies voltage transients larger than 2 V. When such a transient occurs the app checks the machine events identified by the cross correlator to determine which piece of equipment caused the transient. If no events occurred at the



Fig. 9. Identifying large shop loads. The cross correlator identifies transients of machines in the shop that might interfere with the 3-D printer.

```

1 def run(data, state, insert): #pseudo-code
2     for dv in diff(volts):
3         if(abs(dv)>2): #Voltage transient > 2V
4             eqp = find_equipment_transient(dv)
5             if(eqp==None): #no equipment turn-on/off
6                 insert("utility")
7             else
8                 insert(eqp.name)

```

Listing 5. Energy app pseudo code for identifying and assigning responsibility for voltage transients. `find_equipment_transient` is implemented by the cross correlator trained in Fig. 8.

time of the transient, the voltage disturbance is due to an external load not monitored by the NILM. Such information can be used to quantify power quality complaints when negotiating with the utility. The bars on Fig. 10 indicate voltage transients and the colors assign responsibility either a piece of equipment in the machine shop or to the utility in the case of external loads.

While the laser cutter does create a large voltage droop it does so gradually and so does not disturb the printer. The air compressor has much more rapid transients and is identified as an interfering load by the app. There is also a voltage transient that cannot be associated with machines in the shop and the app assigns the transient to the “utility.” In fact this transient was due to a nearby shop vac that, while not physically located in the machine shop, did cause voltage disturbances on the line. This type of disturbance is typical of power quality problems induced by operations “outside” of the facility. The NILM, configured as an energy box, is not only capable of controlling load sequencing within a facility, it is also able to recognize internal versus external power quality offenders.

C. Adding Control to Energy App

As desired, energy apps can also control loads directly using smart plugs. Smart plugs connect to a WiFi network and allow remote clients to control an embedded relay to switch a load on or off. These plugs are available from a variety of vendors [20], [21] but use proprietary protocols that make them difficult to use outside of their private commercial ecosystem. The plug in Fig. 11 is a modified Belkin WeMo Insight. The stock Insight only communicates with a smart phone app and provides limited metering capability. We designed a drop in replacement control PCB that provides the stock functionality

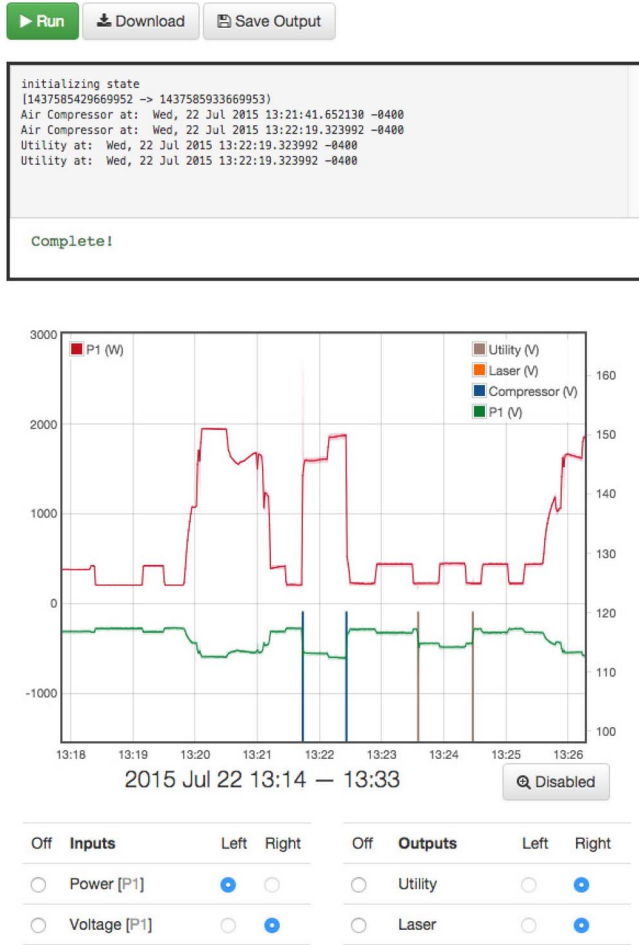


Fig. 10. Identifying the causes of voltage transients. An energy app correlates machine turn-on/off events with voltage transients. If a transient occurred without a matching machine event, the disturbance is assigned to the utility.

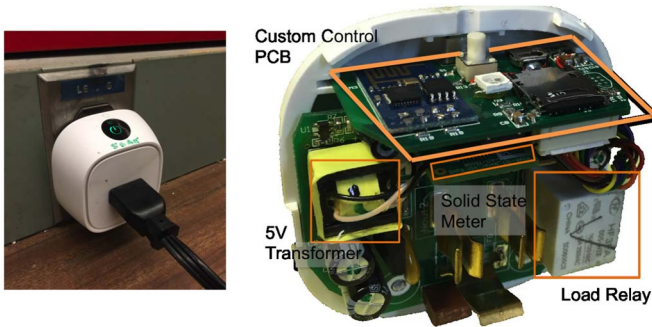


Fig. 11. Custom smart plugs allow the NILM to monitor and control individual loads. This plug is a commercial Belkin WeMo [20] retrofitted with a custom control PCB. Energy apps can control the plug relay and read the embedded solid state meter.

as well as persistent storage to an SD Card, a battery backed real time clock, and high bandwidth metering. This plug interfaces directly with the NILM so energy apps can monitor and control individual loads.

The app outlined in Listing 6 uses one of these smart plugs attached to the air compressor to improve the power quality to the 3-D printer. Fig. 12 shows the result. When the 3-D printer

```
def run(data, state, insert): #pseudo-code
1  if(detect_printer(data)):
2      #printer is running: disable the compressor
3      set_compressor_relay(OFF)
4      last_run = cur_time
5  elif(cur_time-last_run > WAIT_TIME):
6      #printer has been off at least WAIT_TIME:
7      # enable the compressor
8      set_compressor_relay(ON)
```

Listing 6. Energy app pseudo code that only allows the air compressor to run when the 3-D printer is off. The app identifies the 3-D printer by the bed heater waveform and only enables the smart plug relay for the air compressor with the bed has been off WAIT_TIME or longer.

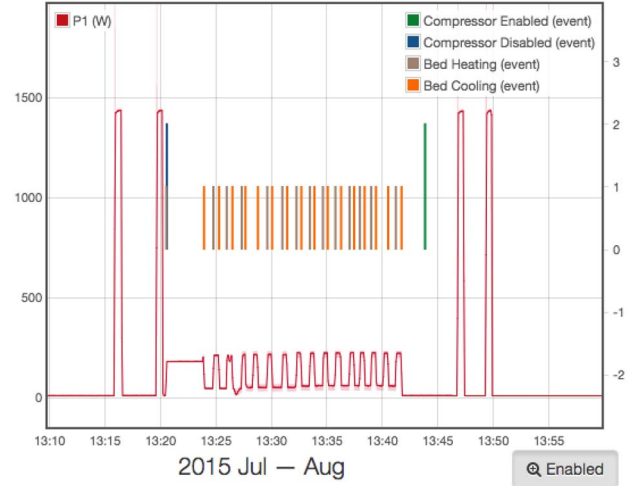


Fig. 12. Power quality app in action: running the code in Listing 6 protects the 3-D printer by disabling the air compressor during print jobs.

turns on (as detected by the cross correlator), the app turns the air compressor off. When the printer has been inactive for at least WAIT_TIME seconds, the compressor is turned back on. The actual compressor runs are determined by a pressure gauge on the machine itself.

VI. CONCLUSION

Detailed energy monitoring and control need not place significant demands on information communication networks. High performance computing makes local analysis, reporting, and reconfiguration inexpensive and precise. Nonintrusive monitoring can provide great capabilities to both utilities and consumers as a mechanism to better understand and control energy consumption; their use and application is no longer limited by the difficulty of viewing and processing their large datasets. NILM manager provides a practical solution for deploying these systems in real world operating environments. The servers described in this paper have been in operation for over a year managing energy monitors across the state of Massachusetts, and as far away as Louisiana. Future work includes scaling the network to support more NILMs, optimizing the data processing framework to run on resource constrained embedded systems, and increasing the privacy and security safeguards which distinguish NILMs from the current smart meter network. NilMDB has been used to integrate consumption data from other utilities like water, and

other diagnostic sources like vibration meters, to provide a comprehensive view of facility operation using NILM manager from anywhere in the world. We envision new service opportunities that can exploit the NILM manager platform to provide energy apps and an “apps market” that grows to satisfy traditional and new demands for resource analytics.

REFERENCES

- [1] Staff Report, Federal Energy Regulatory Commission, “Assessment of demand responses and advanced metering,” Dept. Energy, Fed. Energy Reg. Comm., Washington, DC, USA, Tech. Rep., Dec. 2014. [Online]. Available: <https://www.ferc.gov/legal/staff-reports/2014/demand-response.pdf>
- [2] D. Livengood and R. Larson, “The energy box: Locally automated optimal control of residential electricity usage,” *Service Sci.*, vol. 1, no. 1, pp. 1–16, 2009. [Online]. Available: <http://dx.doi.org/10.1287/serv.1.1.1>
- [3] J. A. Rodríguez-Mondéjar, R. Santodomingo, and C. Brown, “The ADDRESS energy box: Design and implementation,” in *Proc. IEEE Int. Energy Conf. Exhibit. (ENERGYCON)*, Florence, Italy, Sep. 2012, pp. 629–634.
- [4] R. Belhomme, R. Cerero, G. Valtorta, and P. Eyrolles, “The ADDRESS project: Developing active demand in smart power systems integrating renewables,” in *Proc. IEEE Power Energy Soc. Gen. Meeting*, San Diego, CA, USA, Jul. 2011, pp. 1–8.
- [5] S. Althaher, P. Mancarella, and J. Mutale, “Automated demand response from home energy management system under dynamic pricing and power and comfort constraints,” *IEEE Trans. Smart Grid*, vol. 6, no. 4, pp. 1874–1883, Jul. 2015.
- [6] B. Sweet, “The smart meter avalanche,” *IEEE Spectr.*, Oct. 2009. [Online]. Available: <http://spectrum.ieee.org/energywise/energy/renewables/the-smart-meter-avalanche>
- [7] Z. Wang and G. Zheng, “The application of NILM in energy evaluation of smart home: Contrast with IPv6,” in *Proc. Int. Conf. Electr. Inf. Control Eng. (ICEICE)*, Wuhan, China, Apr. 2011, pp. 791–794.
- [8] W. Wichakool, Z. Remscrim, U. A. Orji, and S. B. Leeb, “Smart metering of variable power loads,” *IEEE Trans. Smart Grid*, vol. 6, no. 1, pp. 189–198, Jan. 2015.
- [9] S. R. Shaw, S. B. Leeb, L. K. Norford, and R. W. Cox, “Nonintrusive load monitoring and diagnostics in power systems,” *IEEE Trans. Instrum. Meas.*, vol. 57, no. 7, pp. 1445–1454, Jul. 2008.
- [10] N. Amirach, B. Xerri, B. Borloz, and C. Jauffret, “A new approach for event detection and feature extraction for NILM,” in *Proc. 21st IEEE Int. Conf. Electron. Circuits Syst. (ICECS)*, Marseilles, France, Dec. 2014, pp. 287–290.
- [11] A. N. Milioudis, G. T. Andreou, V. N. Katsanou, K. I. Sgouras, and D. P. Labridis, “Event detection for load disaggregation in smart metering,” in *Proc. 4th IEEE/PES Innov. Smart Grid Technol. Europe (ISGT EUROPE)*, Lyngby, Denmark, Oct. 2013, pp. 1–5.
- [12] M. Dong, P. C. M. Meira, W. Xu, and C. Y. Chung, “Non-intrusive signature extraction for major residential loads,” *IEEE Trans. Smart Grid*, vol. 4, no. 3, pp. 1421–1430, Sep. 2013.
- [13] J. Paris *et al.*, “Scalability of non-intrusive load monitoring for shipboard applications,” in *Proc. Amer. Soc. Naval Eng. Day*, National Harbor, MD, USA, Apr. 2009.
- [14] J. Paris, J. S. Donnal, and S. B. Leeb, “NilmDB: The non-intrusive load monitor database,” *IEEE Trans. Smart Grid*, vol. 5, no. 5, pp. 2459–2467, Sep. 2014.
- [15] Flot. *Attractive Javascript Plotting for JQuery*. Accessed on Jan. 29, 2015. [Online]. Available: <http://www.flotcharts.org/>
- [16] PG&E. *Smartmeter Network—How it Works*. Accessed on Jan. 9, 2015. [Online]. Available: <http://www.pge.com/en/myhome/customerservice/smartmeter/howitworks/index.page>
- [17] ComEd. *Smartmeters: Empowering You to Save Energy and Money*. Accessed on Jan. 9, 2015. [Online]. Available: <https://www.comed.com/Documents/technology/What%20is%20a%20Smart%20Meter.pdf>
- [18] BGE. *How Smart Meters Work*. Accessed on Jan. 9, 2015. [Online]. Available: <http://www.bge.com/smartenergy/smartgrid/smartmeters/Pages/How-Smart-Meters-Work.aspx>
- [19] J. Paris, J. S. Donnal, R. Cox, and S. Leeb, “Hunting cyclic energy wasters,” *IEEE Trans. Smart Grid*, vol. 5, no. 6, pp. 2777–2786, Nov. 2014.
- [20] Belkin. *Wemo Insight Switch*. Accessed on Aug. 28, 2015. [Online]. Available: <http://www.belkin.com/us/p/P-F7C029/>
- [21] DLink. *Wi-Fi Smart Plug*. Accessed on Aug. 28, 2015. [Online]. Available: <http://us.dlink.com/products/connected-home/wi-fi-smart-plug/>



John S. Donnal received the B.S. degree in electrical engineering from Princeton University, Princeton, NJ, USA, in 2007, the M.S. degree in electrical engineering from the Massachusetts Institute of Technology (MIT), Cambridge, MA, USA, in 2013, and is currently working toward the Ph.D. degree at MIT.

His current research interests include nonintrusive load monitoring synthesis, energy harvesting, and communications systems.



James Paris received the B.S., M.S., and Ph.D. degrees in electrical engineering and computer science from the Massachusetts Institute of Technology (MIT), Cambridge, MA, USA, in 2003, 2006, and 2013, respectively.

He is currently with Bolt Innovation Management, Boston, MA, USA. He is currently consulting for hardware startups in the Boston, MA, USA, area.



Steven B. Leeb (S’89–M’91–SM’01–F’07) received the B.S., M.S., and Ph.D. degrees from the Massachusetts Institute of Technology (MIT), Cambridge, MA, USA, in 1987, 1989, and 1993, respectively.

He is currently a Professor with the Laboratory for Electromagnetic and Electronic Systems, MIT. His current research interests include design, analysis, construction, control, and monitoring of servomechanical actuators and mechatronic systems as well as the application of exotic materials, including gel polymers, to actuator construction.