LBNL-XXXXX

ERNEST ORLANDO LAWRENCE
BERKELEY NATIONAL LABORATORY

Development of Diagnostic and M&V Agents, and Implementation in an Occupied Office Environment

Jessica Granderson, Phillip Price, Stephen Czarnecki, Janie Page, Rich Brown, Mary Ann Piette

Energy Technologies Area

March, 2015

This page intentionally left blank

DISCLAIMER

This report was prepared as the result of work sponsored by the California Energy Commission. It does not necessarily represent the views of the Energy Commission, its employees or the State of California. The Energy Commission, the State of California, its employees, contractors and subcontractors make no warrant, express or implied, and assume no legal liability for the information in this report; nor does any party represent that the uses of this information will not infringe upon privately owned rights. This report has not been approved or disapproved by the California Energy Commission nor has the California Energy Commission passed upon the accuracy or adequacy of the information in this report.

Acknowledgement

This work was supported by the Assistant Secretary for Energy Efficiency and Renewable Energy, Building Technologies Office, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

This page intentionally left blank

Glossary of Acronyms 1

Abstract 2

1 Introduction 3

2 System Architecture 4

3 Description of Diagnostic and M&V Agents 6
3.1 Lighting M&V Agent Description 6
3.2 Logic for the Lighting M&V Agent 8
3.3 Diagnostic Agent Description 9
3.4 Logic for the Diagnostic Agent 11

4 Integration of Agents in an Occupied Office Environment 14
4.1 Description of the Office Environment 14
4.2 Integration of the agent platform, lighting control system, and agents 15

5 Measurement and Verification, and Diagnostic Results 18
5.1 Measurement and Verification Agent, Baseline Results 18
5.2 Measurement and Verification Agent, Savings Results 20
5.3 Fault Detection and Diagnostic Results 22
5.3.1 Additional Fault Detection and Diagnostic Results 30
5.3.2 Additional Validation of Fault 1, Lights Don't Turn On/Off According to Programmed Schedule 30
5.3.3 Additional Validation of Fault 2, Lights Don't Turn On/Off According to Manager's Intended Schedule 31
5.3.4 Additional Validation of Fault 3, Schedules Not Optimally Defined 32
5.3.5 Additional Validation of Fault 4, Override Time-outs Not Optimally Defined 33

6 Conclusions and Future Work 34

References 36

Appendices 38
Appendex A.
Psuedocode for the Measurement and Verification Agent 39
Interfacing With the Volttron Message Bus 39
Calculating a Baseline and Quantifying Savings: 40
Appendex B.
Psuedocode for the Fault Detection and Diagnostic Agent 41

Interfacing With the Volttron Message Bus 41
Detecting and Diagnosing Scheduling Faults 43
Appendex C.
Code-level Documentation 45
Purpose 45
System Requirements and Dependencies 46
Additional Considerations and Concepts 46
Installation Instructions 46
Execution Instructions for using standalone modules 46
lighting_baseline Execution Instructions 49
lighting_fdd Execution Instructions 51

TABLE OF FIGURES

Figure 1. Architecture of the Transactive Network, general RTU case [Haack et al. 2013] 5
Figure 2. Architecture of the Transactive Network, lighting implementation 6
Figure 3. Inputs and outputs for the Transactive Network lighting M&V agent 8
Figure 4. Inputs and outputs for the Transactive Network baseline agent 9
Figure 5. Inputs and outputs for the Transactive Network lighting diagnostics agent 11
Figure 6. Diagram of the office environment to demonstrate the lighting Transactive Network 15
Figure 7. Photograph of the office environment to demonstrate the lighting Transactive Network..... 15
Figure 8. Diagram of the integration of the control system, and Transactive Network agent platform 17
Figure 9. Diagram of the base case for the Transactive Network lighting demonstration 18
Figure 10. Plot of the baseline energy profile from the Transactive Network lighting M&V agent 20
Figure 11. Plot of the average power in the base case, scheduled case, as quantified by the Transactive Network lighting M&V agent 21
Figure 12. Plot of the percent savings quantified by the Transactive Network lighting M&V agent, in each zone 21
Figure 13. Graphical representation of faults detected and diagnosed by the diagnostic agent, for Zone 1 25
Figure 14. Graphical representation of faults detected and diagnosed by the diagnostic agent, for Zone 2 26
Figure 15. Data snapshot from two days for which the lights did turn off or come on according to schedule. 27
Figure 16. Data snapshots of two instances when the scheduled off-time occurred before vacancy in Zone 1 28
Figure 17. Data snapshot showing after-hours manual switches to off, before elapse of the override timeout. 29
Figure 18. Data snapshot showing fault-free scheduled operations in Zone 2. 29
Figure 19. Histogram of the occurrence of Fault 1 across 19 scheduled zones at Site 2, for a 4-month period 30
Figure 20. Histogram of the occurrence of Fault 1 across 55 relays at site 3, for a 4-month period 31
Figure 21. Occurrence of Fault 1 by location, over a four-month period at site 2 (left), and over a the three-month period Site 3 (right) 31
Figure 22. Chart showing the difference between the specified and programmed schedules at Site 1 32
Figure 23. Schematic Diagram of Diagnostic and M&V Code 38

Glossary of Acronyms

BAS – Building automation system
BTO – Building Technologies Office
CT – Current transducer
FDD – Fault detection and diagnostics
M&V – Measurement and verification
RTU – Rooftop unit
sMAP – Simple measurement and actuation profile

Abstract

The Building Technologies Office (BTO) at the US Department of Energy is pursuing a framework concept to improve building energy efficiency and increase savings potential in commercial buildings. The "transaction -based framework" enables market-based transactions within and between buildings, and between buildings and the electric grid. Over the past three years, BTO has funded the development of a "Transactive Network" to supports energy, operational and financial transactions. The initial scope focused on transactions between roof top units (RTUs), and those between RTUs and the electric power grid. Using an open architecture agent-based platform called Volttron™ [Haack 2013], a set of applications were developed that reside either on the equipment, on local building controllers, or in the Cloud.

This report documents the development and testing of an extended set of applications, or "agents", for the Tansactive Network, focusing on applications for the lighting end use. Although lighting controls can generally save five to forty percent in end use energy, they remain under-adopted in commercial buildings. Key barriers include high cost and complexity, improper configuration and commissioning that prevent optimal operations and savings, and lack of visibility into achieved savings. In response, this work focuses on two applications - automated fault detection and diagnostics (FDD), and automated measurement and verification (M&V) of energy savings. The agents are designed for schedule-based controls, and in the future, can be enhanced for application to more advanced strategies such as occupancy-based control, and daylighting.

The contents of this report detail the programming of the M&V and FDD applications; specifically, the architecture of the Transactive Network lighting system and associated agents, agent inputs and outputs, and underlying logic. In addition, demonstration and testing results are presented, covering a description of the occupied office environment in which the agents were implemented, energy use characterization for the no-controls-automation base case, achieved energy savings, and results from executing the diagnostic agent.

The ability to effectively and accurately quantify *project specific* savings was successfully demonstrated for the M&V agent. This is in contrast to demonstration studies that have focused on establishing *general savings levels* that can be expected from one control strategy versus another. For example, it is generally accepted that scheduling can save five to fifteen percent in energy use. However, given the atypical occupant behavior in the demonstration site, the measured and verified savings were on the order of sixty percent. These savings are significantly higher than those generally expected when implementing schedule-based controls. This was due to the fact that energy use in the base case was extremely high because of severe under-use of manual controls; and occupants commonly left the lights on all night and all weekend in both demonstration zones. The ability of the FDD agent to successfully identify scheduling faults was also demonstrated in the Transactive Network demonstration site, and with data from an additional set of buildings with schedule-based controls.

1 Introduction

[Somasundaram et al. 2014] The Building Technologies Office (BTO) at the US Department of Energy is pursuing a framework concept to improve building energy efficiency and increase savings potential. The "transaction-based framework" enables market-based transactions within and between buildings, and between buildings and the electric grid. Four types of transactions are envisioned: end user services; energy market services; grid services; and societal services. Within this framework, transaction-based building controls offer the potential to generate new value streams by negotiating complex interactions, and market-based contracts in parallel with, or instead of conventional control paradigms.

Over the past three years, BTO has funded the development of a "Transactive Network" to supports energy, operational and financial transactions. The initial scope focused on transactions between roof top units (RTUs), and those between RTUs and the electric power grid. Using an open architecture agent-based platform called Volttron™ [Haack 2013], a set of applications that reside either on the equipment, on local building controllers, or in the Cloud, were developed. These agents were developed through a multi-laboratory collaboration between Pacific Northwest National Laboratory, Lawrence Berkeley National Laboratory (LBNL), and Oak Ridge National Laboratory. Key applications included automated fault detection and diagnostics for RTUs, advanced control strategies for RTUs, continuous measurement and verification of energy savings, renewable integration, refrigeration, and demand response [Katipamula 2013; Piette 2013; ORNL 2014].

This report documents the development and testing of an extended set of agents for the Transactive Network, focusing on applications for the lighting end use. Although lighting controls can generally save five to forty percent in end-use energy [Williams 2011], they remain under-adopted in commercial buildings. Key barriers include high cost and complexity, improper configuration and commissioning that prevent optimal operations and savings, and lack of visibility into achieved savings. In response this work focuses on two applications: 1) automated fault detection and diagnostics, and 2) continuous measurement and verification of end use energy savings. The agents are designed for schedule-based controls, and in the future can be enhanced for application to more advanced strategies such as occupancy-based control, and daylighting.

Sections 2 and 3 of this report contain documentation associated with the programming of the agents, including:

- the architecture of the Transactive Network lighting system and associated agents
- a description of the agents that were programmed, including inputs and outputs, and the logic underlying the algorithms

Sections 4-5 contain details of how the agents were integrated into the occupied office environment, and testing and implementation results, including:

- description of the office environment
- integration of the Volttron agent platform, lighting control system, and agents
- baseline energy use characterization of the base case (manual occupant switching only, no controls automation) implemented for the M&V agent
- achieved energy savings relative to the base case
- results from executing the diagnostic agent

Section 6 provides a conclusion and summary of next steps.

2 System Architecture

Figure 1 shows the generalized architecture of the RTU Transactive network system that was developed in prior work. This is provided as a reference point to understand the lighting-focused architecture that was developed in this effort. A Modbus device interface connects the local RTU controller to the agents, and the local controller operates the rooftop unit. Data are stored in the cloud, and the on-site Volttron instance is used to message between applications, data stores, and devices.

Shown in Figure 2, Transactive lighting system architecture includes a diagnostic agent, and an M&V agent that reside within the Volttron platform. Analogous to the prior RTU network architecture, a BACnet device interface links the local lighting controller to the agents, and the local controller actuates the lights. Data relevant to the control, diagnosis, and savings M&V are stored in the cloud, using the sMAP protocol [UC Berkeley 2014], and the on-site Volttron instance is used for messaging.

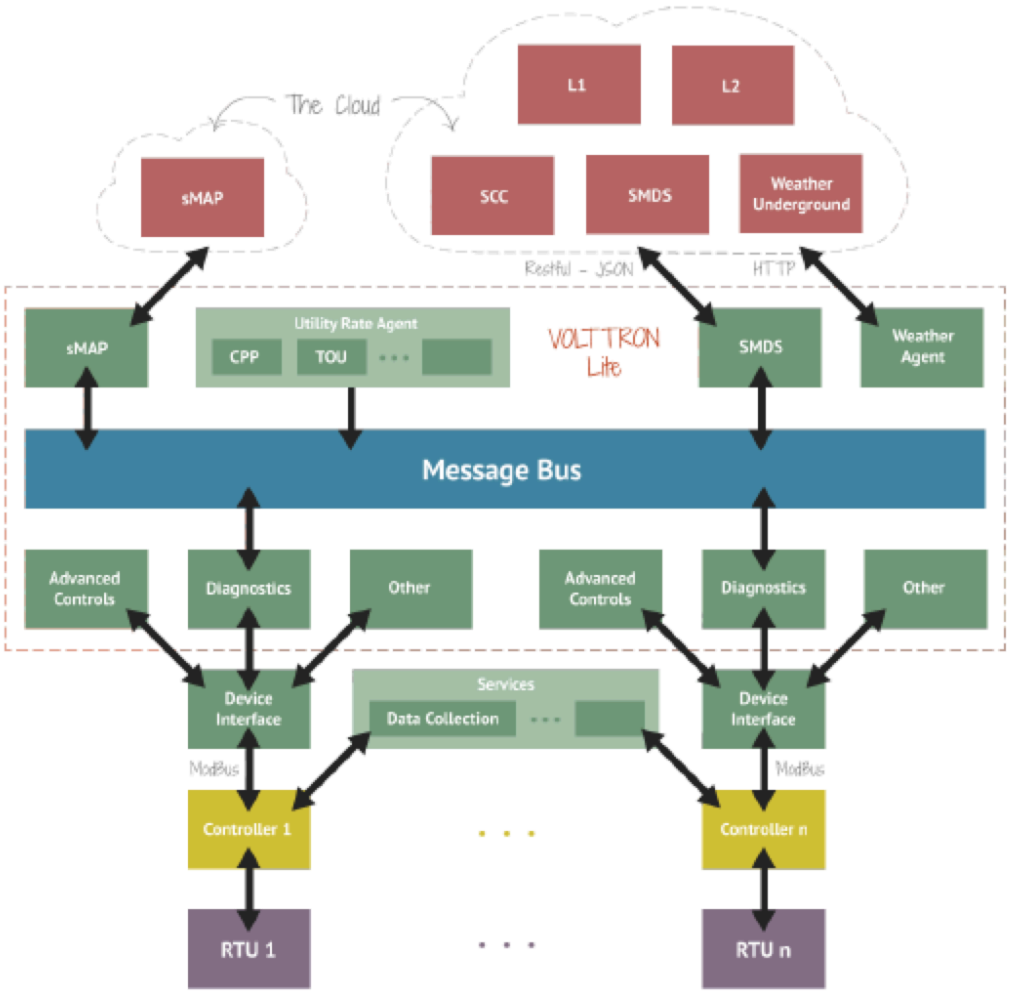

Figure 1. Architecture of the Transactive Network, general RTU case [Haack et al. 2013]

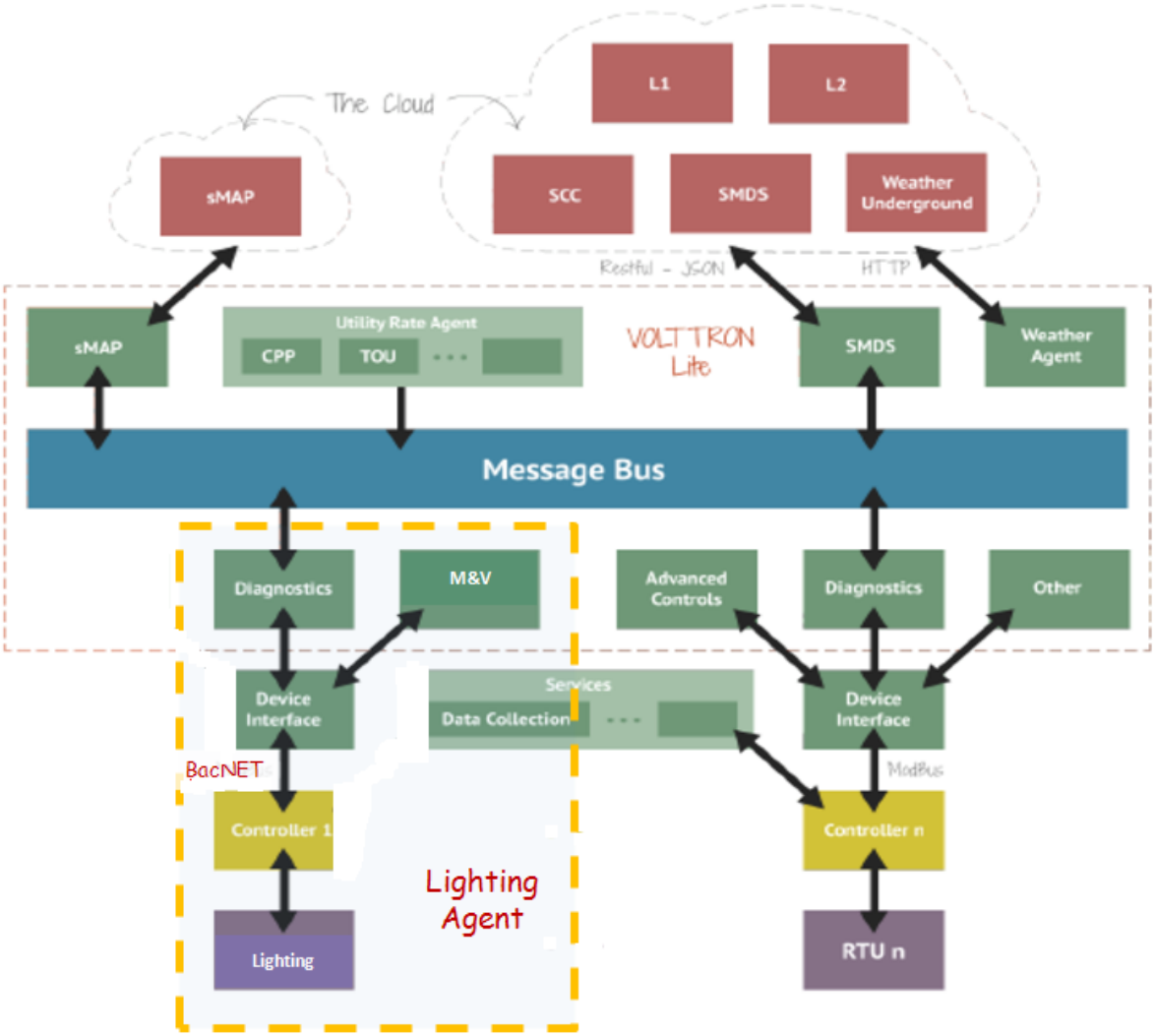

Figure 2. Architecture of the Transactive Network, lighting implementation

3 Description of Diagnostic and M&V Agents

3.1 Lighting M&V Agent Description

The lighting M&V agent quantifies the savings associated with the implementation of automated control strategies, relative to a base case of no controls automation. A recent study has shown that 70% of all commercial buildings, and 52% of all offices feature no automated lighting controls [NCI 2012]. This relatively low level of adoption presents a significant opportunity for concerted efforts to accelerate deployment by addressing barriers associated with perceptions of technology effectiveness in generating reliable, cost effective energy savings.

While implemented in this project to quantify savings associated with scheduling, the agent

design follows the general M&V case. It quantifies the change in lighting energy consumption between two time periods, and therefore can be used to compare the energy used before and after an Energy Conservation Measure is installed. Data from the "pre-installation" or simply the "pre" period, are compared to those from the post-installation or "post" period.

Inputs and outputs to the lighting M&V agent are shown in Figure 3, and summarized below. The outputs of the agent can be accessed by other applications with user interfaces, such as web-based visualization and tracking tools.

Inputs:

1. Lighting load interval data from "pre" period
2. Lighting load interval data from "post" period
3. List of holiday dates (optional)
4. Hour of day that defines the start of the 12-hour "day" for purposes of summarizing changes by daytime and nighttime (optional; the default is 6:30 a.m.)

Outputs:

1. Time series of difference in load between pre and post period, for an average week of each
2. Summary of average load difference by {weekend/weekday/holiday} and {daytime/nighttime}.
3. Average load reduction for an average week with no holiday

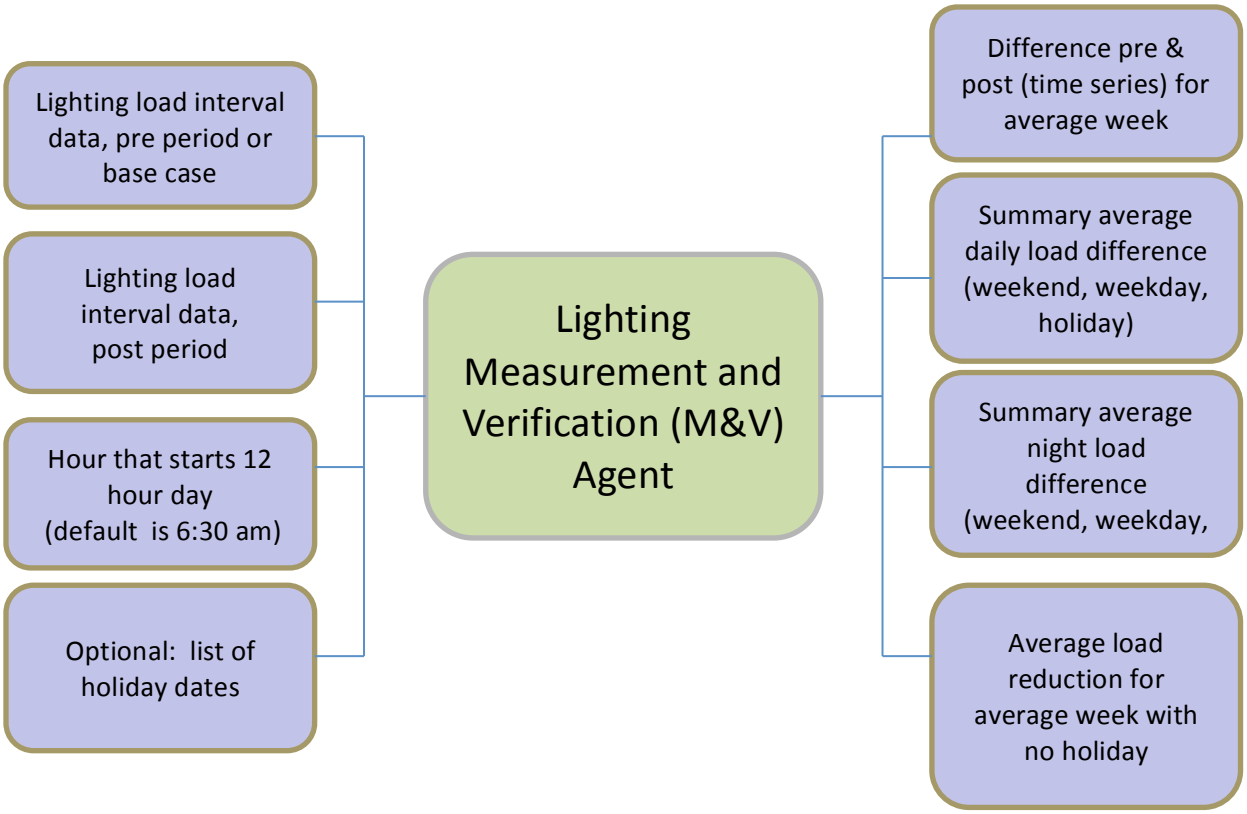

Figure 3. Inputs and outputs for the Transactive Network lighting M&V agent

3.2 Logic for the Lighting M&V Agent

The logic underlying the overall steps in the M&V algorithm is provided below. Detailed pseudocode sufficient for programmers to replicate the algorithm in a programming language other than Python is included in the Appendix.

The lighting M&V agent first calls a separate "lighting baseline agent" that analyzes lighting data from the pre period in order to generate a baseline prediction for the post period. The baseline agent uses very simple logic: its prediction for a given time during the week on a non-holiday is the average load at that time of the week for other non-holidays during the pre period; for example, the prediction for a Tuesday at 10:15 a.m. is the average load at 10:15 a.m. across all Tuesdays. For a holiday, the day of the week is ignored: the prediction for a holiday at 10:15 a.m. is equal to the average load on 10:15 a.m. during holidays. A complication is how to handle the case that the pre period does not contain a holiday, but the post period does; in this version of the software we treat the holiday as a typical day, ignoring the fact that it is a holiday, however other default behaviors can be configured.

The lighting baseline agent can be thought of as a module within the M&V agent, but is implemented as a standalone agent that can be called independently of the M&V agent if desired. Inputs and outputs to the baseline agent are shown in Figure 4, and summarized below. As for the M&V agent, the outputs of the diagnostic agent can be accessed by other

applications with user interfaces, such as web-based visualization and tracking tools.

Inputs:

1. Time series of load data during “pre” period
2. Timestamps for which predictions are desired (typically the “post” period)
3. List of holiday dates (optional)

Outputs:

1. Time series of predicted load during the specified period

As described in Section 3.1, after obtaining a baseline prediction from the Baseline Agent, the M&V agent compares the baseline prediction to the actual load in the post period, and summarizes the differences in several ways including: the difference in average load on weekend days and nights, weekday day and nights, and holiday day and nights; and the resulting difference in energy consumption for an average week that does not include any holidays. The latter quantity is the main quantity of interest but the others may be useful in understanding the result.

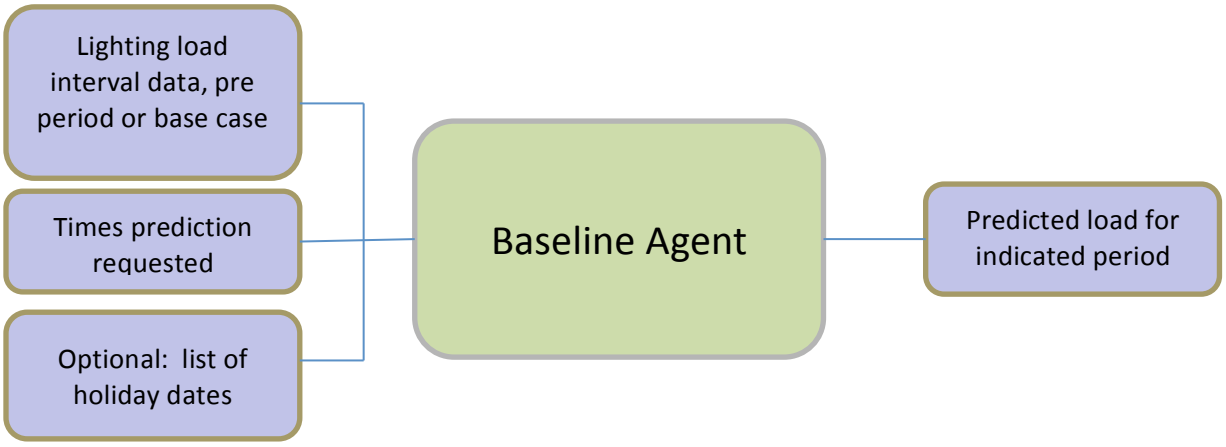

Figure 4. Inputs and outputs for the Transactive Network baseline agent

3.3 Diagnostic Agent Description

The diagnostic agent addresses four faults associated with scheduled lighting controls that can negatively impact energy savings and occupant experience:

1. Lights don’t turn on/off according to programmed schedule times
2. Lights don’t turn on/off according to building manager’s intended schedule,
3. Schedules are not optimally defined,
4. Overrides are not optimally configured

The inputs and outputs of the diagnostic agent are shown in Figure 5. These inputs represent

the superset of possible diagnostic data, however in any given building, it is likely that only a subset of inputs will be available. The algorithms were designed to provide useful outputs, even when provided a subset of the potential inputs. If accessible to parallel applications through, for example BACnet or other control system interfaces, the implemented schedule can be acquired by the FDD program; otherwise it is configured upon installation of the program. Relay status, lighting load, and occupancy are also data streams that may be accessible through BACnet or other interfaces.

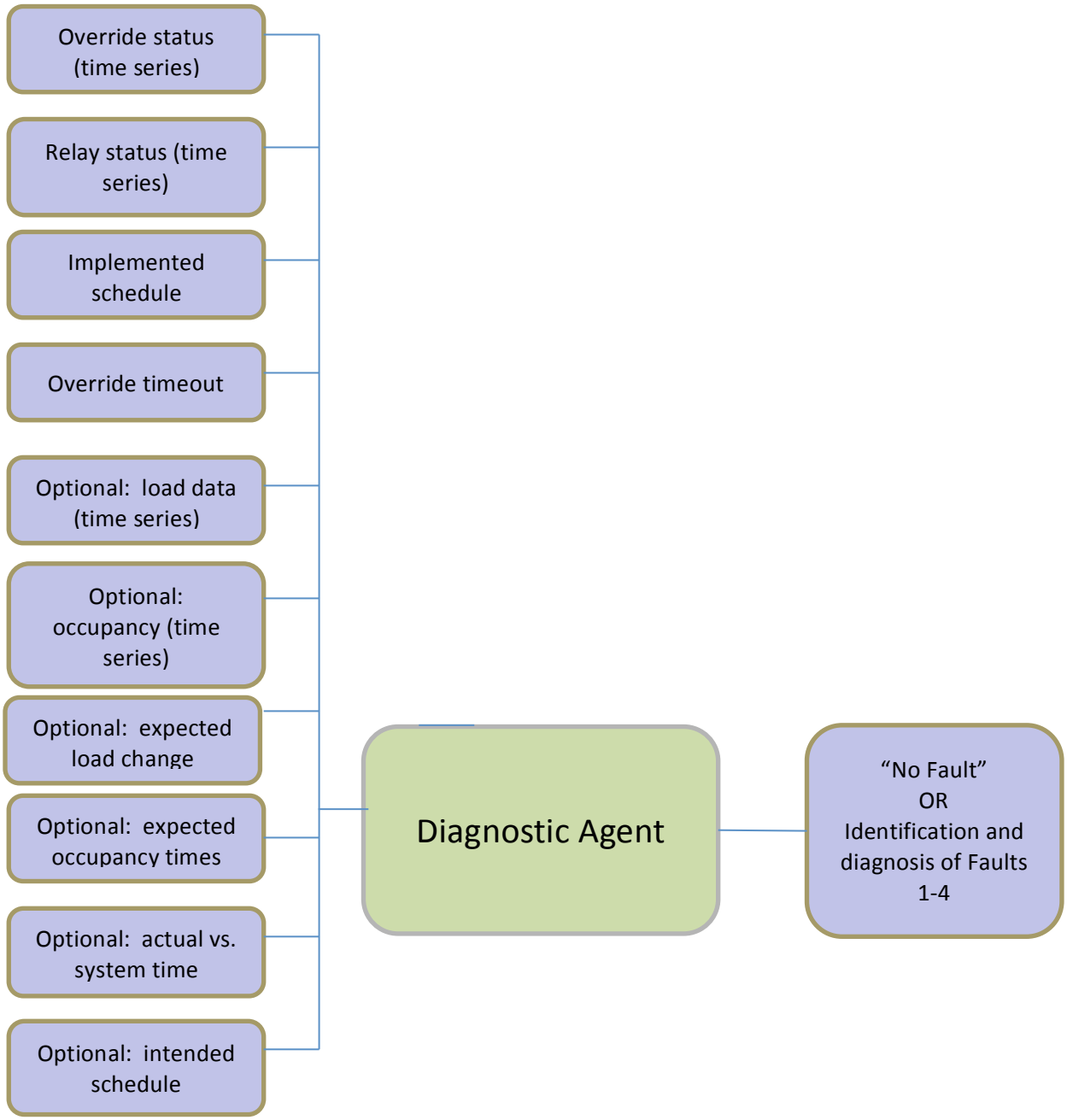

Figure 5. Inputs and outputs for the Transactive Network lighting diagnostics agent

3.4 Logic for the Diagnostic Agent

The logic that describes the overall steps in the diagnostic algorithm is provided below. [Detailed pseudocode sufficient for programmers to replicate the algorithm in a programming language other than python is included in the Appendix.]

Fault 1: Lights don't turn on/off according to programmed schedule times
The following "checks" underlie the logic used to detect and diagnose cases in which the lights do not turn on or off at the times dictated in the programmed schedule:

- Is the system time correct?
- Does relay/group status match the on/off states programmed in the schedules?
- Did a switch event (occupant override) occur?
    - Did the switch event cause the mismatch between relay status and programmed schedule?
- If load data is available, do loads change as expected at scheduled times?

The answers to each of these 'checks' are combined in the logic to determine whether the fault has occurred, and to identify potential causes, or system failures. System failures associated with Fault 1 include: time clock failures, command logging errors, communication failures, and actuator failures.

Fault 2: Lights don't turn on/off according to manager's intended schedule
The following "checks" underlie the logic used to detect and diagnose cases in which the lights do not turn on or off according to the manager's intended schedule.

- Do programmed on/off times all for weekends, weekdays and holidays equal those specified by the building manager?
    - Schedules specified by the building manager comprise a priori data
- If schedules are correctly programmed, and match the manager's intent, do lights turn on/off according to those programmed schedules?

The answers to each of these 'checks' are combined in the logic to determine whether the fault has occurred, and to identify potential causes, or system failures. System failures associated with Fault 2 include: programming errors, and by association with Fault 1, time clock failures, command logging errors, communication failures, and actuator failures.

Fault 3: Schedules are not optimally defined
Determining optimal scheduling is a more fuzzy or qualitative problem than that of diagnosing appropriate time-of-day actuation, as in Faults 1 and 2. For the purposes of this project *suboptimal* is defined as conservative scheduling that presents efficiency opportunities, or as overly-aggressive scheduling that delivers poor service to the occupants. Given that the fault is

defined more qualitatively, the algorithm flags the potential to improve operations or efficiency rather than detecting and diagnosing failures in system operations.

If any of the following are true, flag a potential opportunity to improve efficiency by reducing scheduled hours-on:

- Do programmed on/off times align with expected building occupancy?
    - Are programmed 'on' times much earlier than expected occupancy?
    - Are programmed 'off' times much later than expected vacancy?
    - Expected occupancy and vacancy times comprise a priori data
- Is duration of scheduled 'off' time too short on weekdays or on weekends?
- Do switch events to 'off' occur frequently and prior to scheduled 'off' times?
- If zone occupancy data is available
    - Is the time of first morning entry fairly regular (low variability), and is the average time of first morning much later than scheduled on times?
    - Is the time of last evening exit fairly regular (low variability), and is the average time of last exit much earlier than scheduled off times?

If any of the following are true, flag a potential opportunity to improve service by extending the scheduled hours-on:

- If the system is auto-on (vs. manual-on)
    - Do switch events to 'on' occur frequently and soon after scheduled 'off' times?
    - Do switch events to 'on' occur frequently and prior to scheduled 'on' times?
- If zone occupancy data is available
    - Is the time of first morning entry fairly regular (low variability), and is the average time of first morning much earlier than scheduled on times?
    - Is the time of last evening exit fairly regular (low variability), and is the average time of last exit much later than scheduled off times?

Fault 4: Overrides are not optimally configured

Similar to the case of optimal scheduling, optimal configuration of overrides is a fuzzier problem than that of diagnosing appropriate time-of-day actuation. For the purposes of this project *suboptimal* is defined as time-out configurations that are too long, and present efficiency opportunities, or that are too short, and result in poor service for the occupants. Given that the fault is defined more qualitatively, the algorithm flags the potential to improve operations or efficiency rather than detecting and diagnosing failures in system operations.

If any of the following are true, flag a potential opportunity to improve efficiency by reducing the length of the override time-out:

- Do switch events to 'off' occur frequently and prior to elapse of the time-out?
- If zone occupancy data is available
    - Is duration of after-hours occupancy events significantly shorter than the length of the time-out?

If any of the following are true, flag a potential opportunity to improve service by increasing the length of the override time-out:

- Do switch events to 'on' occur frequently and soon after the time-out has elapsed?
- If zone occupancy data is available
    - Is duration of after-hours occupancy events significantly longer than the length of the time-out?

4 Integration of Agents in an Occupied Office Environment

In this section we describe the office environment into which the Transactive Network lighting agents were integrated. The office layout, zoning, lighting system, and occupant characteristics are overviewed. In addition we detail the agents' configuration, their hosting on the Volttron platform, and the communication pathways and data streams that integrate the office lighting system and agents.

4.1 Description of the Office Environment

The environment that was used for this work consists of open-plan cubicle spaces, in the interior of an office floor that includes perimeter personal office spaces. This floor comprises the occupied 'Lighting and Plug Load Testbed' in LBNL's FLEXLAB [LBNL 2014a]. The Transactive Network demonstration was conducted across 17 occupant cubicles, configured into two lighting control zones, as shown in Figure 6 (zone boundaries are indicated in Figure 9). Each zone features two fixture banks, and each bank contains six linear fluorescent lamp segments that are oriented vertically with respect to the diagram in Figure 6. These fixtures can be controlled manually, via occupant-accessible switches, or programmed for automated control, through a WattStopper DLM (Digital Lighting Management) system [WattStopper 2014]. Figure 7 contains a photograph of this demonstration space. The occupants in these office spaces are researchers, who typically conduct a mix of computer- and paper-based work.

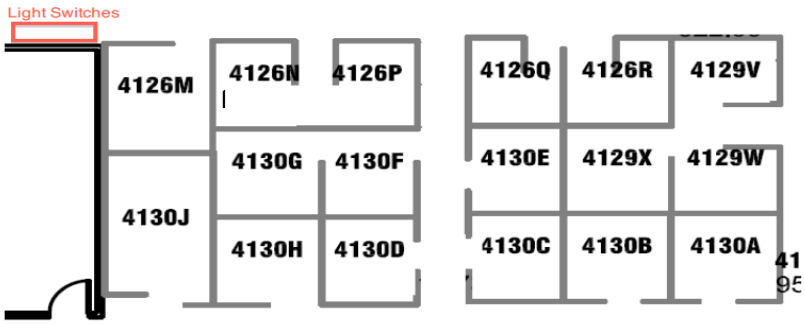

Figure 6. Diagram of the office environment to demonstrate the lighting Transactive Network

Figure 7. Photograph of the office environment to demonstrate the lighting Transactive Network

4.2 Integration of the agent platform, lighting control system, and agents

Figure 2 in Section 2 of this report provided a schematic block diagram of the Transactive Network lighting implementation. Here, we describe in further detail, the integration of the agents, with the host Volttron platform, and the office lighting system, focusing on the software components, their location on diverse computer hardware, and data communications and storage elements of the integrated system. In this implementation the agents and database were locally on an on-site computers, however, they can also be located in the cloud.

At the highest conceptual level, the system performs the following functions: (1) monitor the lighting in two zones, including operation of switches and relays as well as the power consumption; (2) log the data to a sMAP database; (3) run software agents to analyze the data and report results, with the communication between agents and the database taking place via the Volttron message bus.

Figure 8 shows a schematic diagram of the monitoring and control of the lighting system, and integration with the Transactive Network platform and agents. The Wattstopper DLM system is the hardware and software infrastructure for sensor data acquisition and control of the lights. There are six overhead lamp segments in each zone. Each light is plugged into an outlet that is connected to an LMRC-211 Single Relay w/0-10V Dimming Room Controller. The LMRC-211 is connected to a LMBC-300 Network Bridge, along with other plug load controllers, switches, occupancy sensors and light sensors located within a single office. Up to five network bridges are daisy chained together to form a single BACnet MS/TP segment network and connected to a BaS (BACnet MS/TP to BACnetIP) router. The BaS router is then connected to a regular Ethernet switch. Each router has a maximum of six network segments and coordination of lighting controllers is limited and isolated to each router.

Since the lights in each zone span multiple routers, the Wattstopper Segment Manager efficiently synchronizes controls to simulate zone controls. For the base case, all six lights in each zone are associated with a single digital light switch for manual on-off operation. For the scheduled control case, a schedule can be assigned to the zone lights and manual override settings can be assigned to the light switch. For the occupancy-based control case, the zone lights will be controlled with feedback from the zone occupancy sensors. All lighting controls for each case are programmed directly into the Wattstopper DLM.

Power measurements of the lights are converted through a National Instruments (NI) compact Realtime Input Output (cRIO) system to be read over Ethernet. Each 120V circuit for lighting has current transducers (CTs) that output a low voltage signal that varies in proportion to the amount of current flowing through the main circuit. These low voltage CT signals are individually fed via BNC connections to a patch panel that is interfaced to the NI cRIO chassis. The cRIO converts the analog CT signals to digital values that can be retrieved over Ethernet.

For network security reasons, all communication with the Wattstopper and National Instruments devices is through the Calbay Central Workstation (CWS) server portal. The CWS communicates over Ethernet to the BaS routers to monitor and control all Wattstopper devices using the BACNet protocol.

The sMAP client polls the CWS every five minutes to fetch real-time load data, as well as Wattstopper device states, including light switches, relay states, and occupancy sensor readings. The retrieved data is posted to smap.lbl.gov for archiving and visualization purposes.

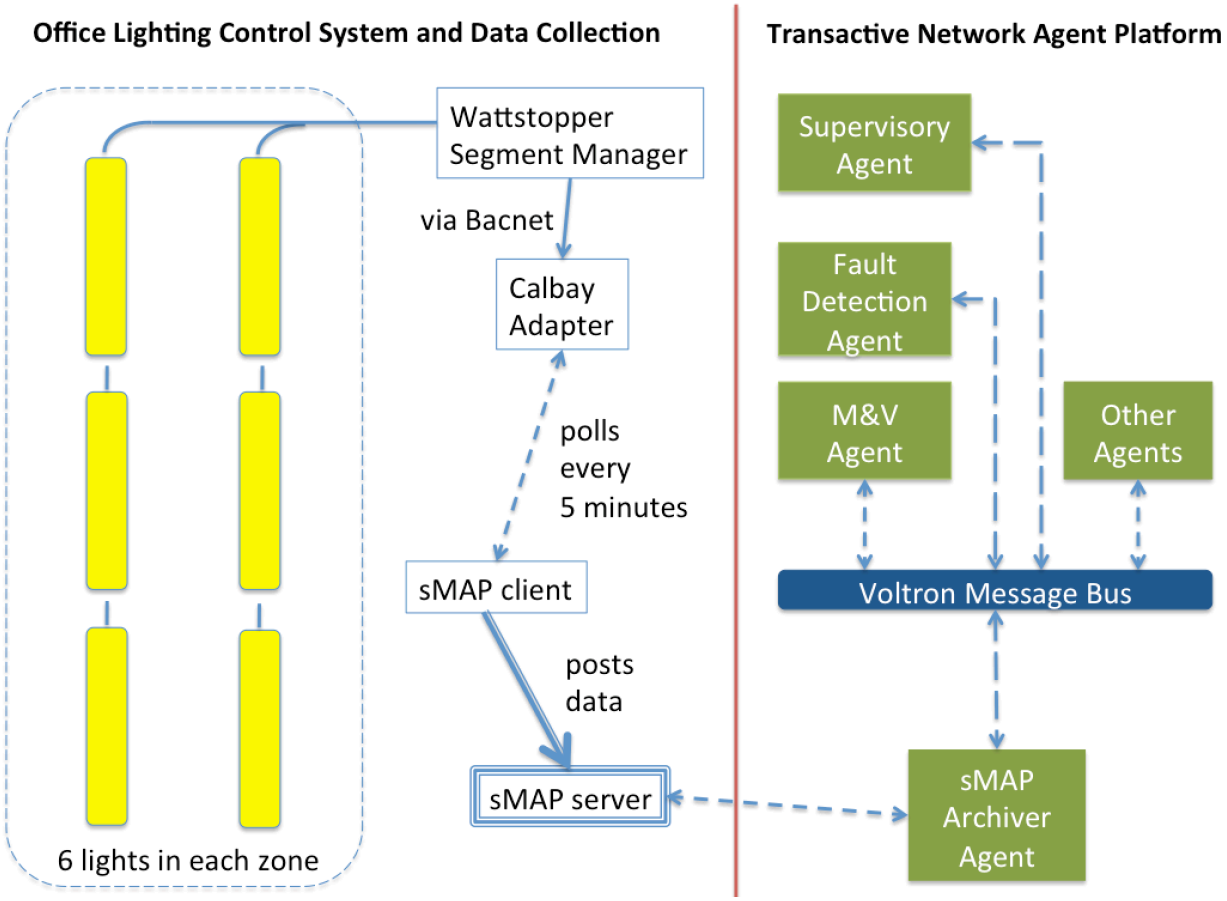

Figure 8. Diagram of the integration of the control system, and Transactive Network agent platform

The workflow of the data analysis system is as follows.

1. A "Supervisory Agent" is run. One of its tasks is to schedule itself (via the PNNL Scheduler Agent) to run again one minute later.
2. The Supervisory Agent requests data from sMAP: zone power, relay state, switch state.
3. When the requested data are provided, the Supervisory Agent calls the M&V Agent and the Fault Detection Agent and sends them the appropriate data via the Volttron Message Bus.
4. Each agent operates on the input data and sends the results back to the Supervisory Agent via the Volttron Message Bus.
5. The Supervisory Agent reads the results from the Message Bus and writes them to local files.

The diagnostic and measurement and verification agents have been submitted to the repository at https://bitbucket.org/berkeleylab/eetd-tn-lighting. This is a private repository where collaborators can be granted access as needed.

5 Measurement and Verification, and Diagnostic Results

Following integration of the agents into the demonstration office environment, a base case was implemented that represents the most common case in offices – manual, switch-based occupant control, with no automation [NCI 2010]. The base case was designed and configured to reflect the most common office implementations. Specifically, the lights in each zone (the 'Blue Zone' and the 'Gold Zone') are commonly actuated to 'on' and 'off' states with a wall-mounted switch accessible to the occupants. This base case configuration is shown in Figure 9. The base case was implemented for 33 days (almost 5 weeks), and used to establish the baseline energy consumption model that was used in the M&V agent. The M&V agent is described in Sections 3.1 and 3.2 of this report.

Figure 9. Diagram of the base case for the Transactive Network lighting demonstration

Once the baseline energy consumption model was established, the fault detection and diagnostic agent was executed to verify that the agent was able to correctly identify the absence or incorrect implementation of schedules.

5.1 Measurement and Verification Agent, Baseline Results

The baseline model used for the M&V agent was used to quantify the energy savings associated with the implementation of a schedule-based control strategy. In the Blue Zone (Zone 1) the schedule was set to "on" at 8:00 AM, and "off" at 6:00 PM. In the Yellow Zone (Zone 2), the schedule was set to "on" at 6:00 AM, and "off" at 6:00 PM. The slight difference in schedules was to facilitate testing and validation of system configurations as well as algorithm and sensed data outputs – not to match any known differences in the occupant's actual occupancy patterns. In both zones, the timeout for after-hours occupant overrides from "off" to "on" was configured to 2 hours.

As described in Section 3.2, the M&V baseline model characterizes the average lighting weekly lighting load for a manual controls base case, for {weekends/weekdays/holidays}, and during {daytime/nighttime} hours. The inputs for the baseline model include the load time series and, optionally: a start time for the "daytime" (default 6:30 a.m.); a list of which days are "workdays" (default Monday – Friday); and list of holiday dates.

Table 1 summarizes the specific data streams that were available from the demonstration office environment, and used to create the baseline model. Zone power may be available in cases where scheduling is implemented using dedicated lighting automation panels, or stand-alone lighting control systems that offer ballast control; in cases where submetered power data is not available, relay status can be combined with knowledge of the controlled lighting load (kW), to create a calculated measure of power.

Table 1. M&V baseline model inputs, data sources, and measurement and calculation details

Model input	Source of data	Measurement and calculation details
Time series of zone power	Wattstopper data, logged to sMAP	'Virtual' sensor point created by summing power data from individual ballasts. 'Virtual' sensor point created from relay data from individual ballasts, multiplied by lamp load information
Holiday and weekend/weekday dates	Standard US calendar	
'Daytime' and 'Nighttime' hours	Hardcoded algorithm input	A 12-hour day is assumed, with a default daytime start defined as 6:30 AM, nighttime start at 6:30 PM

The Lighting M&V Agent was demonstrated using data from the occupied office environment. Table 2 summarizes the baseline energy use for the base case, which was then used to quantify savings from the use of scheduling.

Table 2: Baseline energy profile from the base case, quantified by the Transactive Network lighting M&V agent

Day Type	Daytime (6 a.m. – 6 p.m.)		Nighttime (all other times)	
	Blue Zone	Gold Zone	Blue Zone	Gold Zone
Workday	688 W	691 W	673 W	676 W
Non-Workday	709 W	712 W	708 W	708 W
Holiday	No data	No data	No data	No data

Total average power consumption for a non-holiday week: Blue Zone (Zone 1) 687 W, Gold Zone (Zone 2) 691 W.

To help visualize the results shown in the table, Figure 10 contains a plot of the baseline energy use that characterized the base case, as monitored over the 33-day measurement period. Mean load is shown for each time period during the week, and the average load during the week is shown at the far right. The data reflect the actual manual switching behaviors from the office occupants. In order to capture the most realistic energy usage conditions, no intervention or

attempt to influence occupant behavior was made by the research team. As the figure indicates, lights in both zones were left on almost all the time in the baseline period.

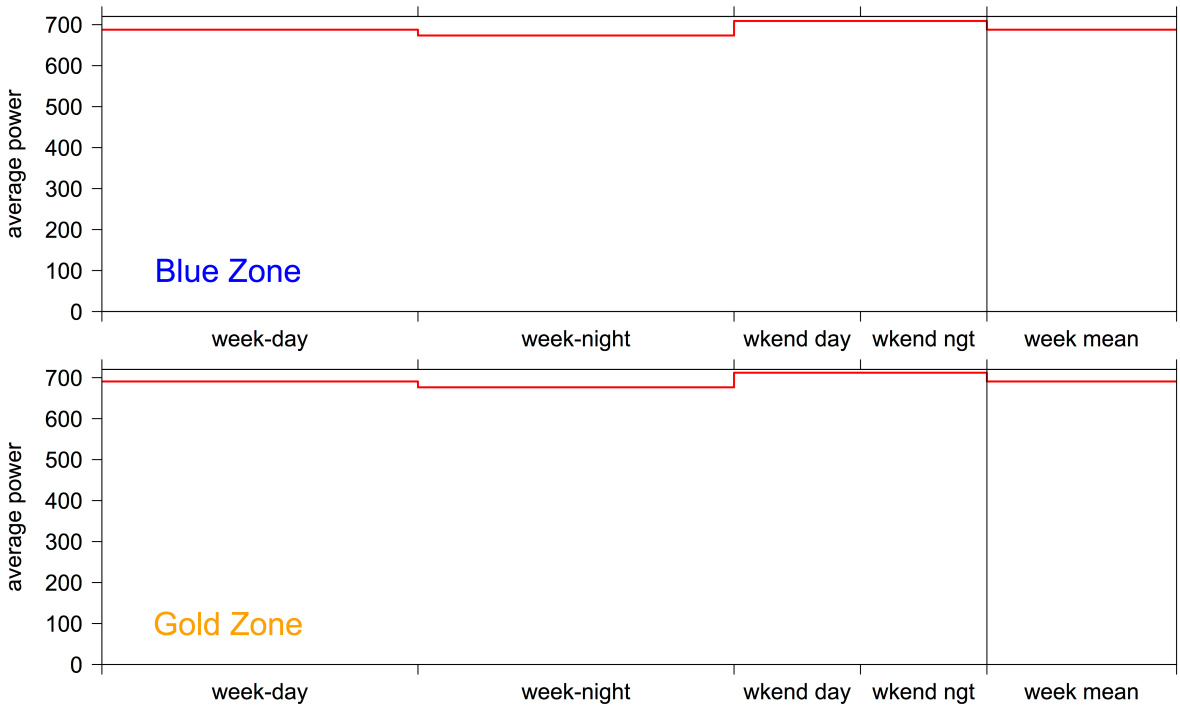

Figure 10. Plot of the baseline energy profile from the Transactive Network lighting M&V agent

5.2 Measurement and Verification Agent, Savings Results

The energy savings that resulted from the implementation of schedule-based controls in each zone were significant. Figure 11 shows the energy savings that were achieved relative to the base case over a 9-week monitoring period, after incorrect scheduling configurations were detected and resolved (see Fault Detection and Diagnostic Results, Section 5.3).

As the figure shows, very large changes in lighting power consumption were observed. In contrast to the baseline case, in the scheduled case the lights were off in both zones for almost all of the time during weekends, and for most of the time on weeknights. The result is a large reduction in weekly average load, shown at the right side of the plot: Weekly mean lighting load in the Blue Zone (Zone 1) went from 688 W to 248 W, and in the Gold Zone (Zone 2) from 691 W to 277 W.

Figure 12 shows the percent energy savings for each time category during the week, and for the average non-holiday week. Compared to the baseline period, scheduled controls saved 64% of lighting energy in the Blue Zone (Zone 1) and 59% in the Gold Zone (Zone 2).

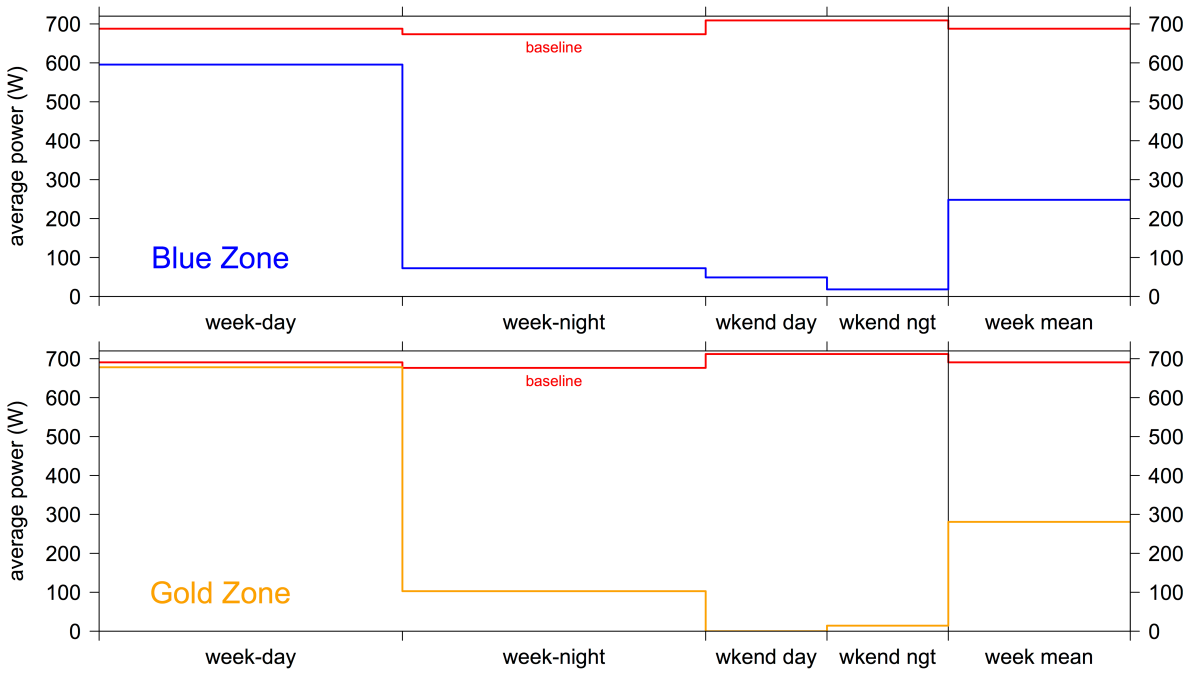

Figure 11. Plot of the average power in the base case, scheduled case, as quantified by the Transactive Network lighting M&V agent

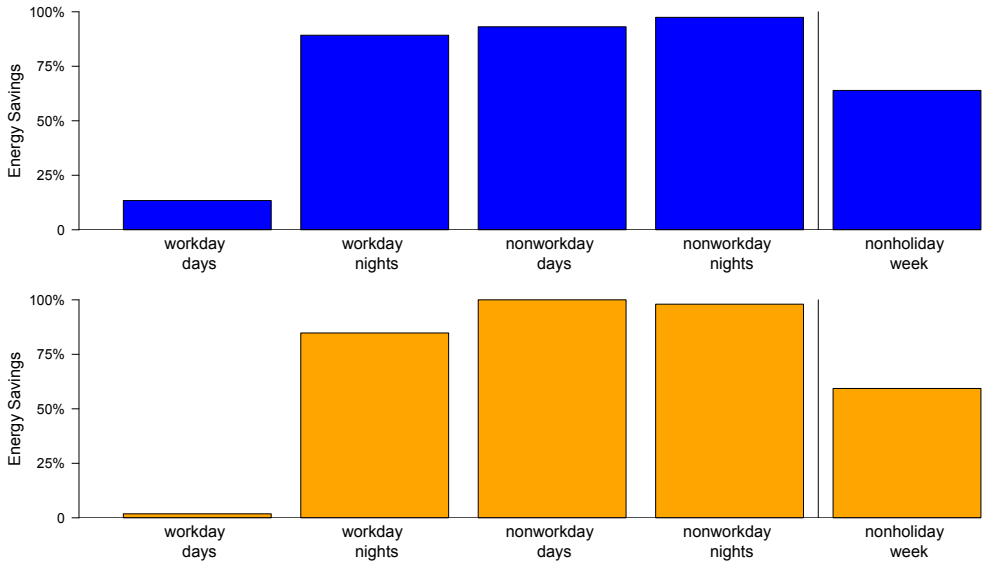

Figure 12. Plot of the percent savings quantified by the Transactive Network lighting M&V agent, in each zone

Throughout the base case, occupants in both zones of the open-plan test space did not acquire a habit of turning the lights off when the last occupant left for the night. The lights were left on all night, and all weekend. This is common in shared open-plan spaces where no single person 'owns' the common switch for the zone, and where individuals may be unaware that they are the last person to vacate the space. However, the office space that was used for this work comprises a 'living laboratory' in which highly granular, workstation-specific automated controls were previously the norm. Therefore, the occupants were even less likely to turn the zone lighting off at night, when compared to a typical office space. As a result, the energy savings that were achieved in demonstrating the M&V and diagnostic agents are significantly higher than in a standard, more general case. This is an acceptable result, given that the purpose of this project was to demonstrate the Transactive network lighting agents, not to establish generalizable savings associated with scheduling. In general, commonly cited expected savings from scheduling range from 5-15%.

5.3 Fault Detection and Diagnostic Results

Following integration of the diagnostic agent into the demonstration space, the agent was executed first for the base case, and then in parallel with the schedule-based controls. Since the base case did not include schedule-based controls, it was possible to verify that the agent was able to correctly identify cases in which: 1) the lights don't turn on/off according to programmed schedule times; and 2) the lights don't turn on/off according to a building manager's intended schedule. Once the agent was validated with data from the base case, it was run in combination with the schedule-based controls.

As described in Section 3.3, the diagnostic agent is capable of handling diverse inputs, depending on the data that is available in the specific building in which it is implemented. However, the algorithms were designed to provide useful outputs, even when provided a subset of the many potential inputs. Table 3 summarizes the specific data streams that were available from the demonstration office environment.

Table 3. Diagnostic agent inputs, data sources, and measurement and calculation details

Model input	Source of data	Measurement and calculation details
Time series of zone lighting on/off status	WattStopper data, logged to sMAP	Direct measure of zone (off/on) relay status.
Time series of occupant switch events	WattStopper data, logged to sMAP	Direct measure of zone switch events (binary).
Programmed schedule	Control system input/configuration settings	Blue Zone: On at 8am off at 6pm every day. Gold Zone: On at 6am off at 6pm every day.
Time series of zone power.	WattStopper data, logged to sMAP	'Virtual' sensor point created by summing power data from individual ballasts.
Expected load change	Hardcoded for testing purposes.	Minimum and maximum zone power change upon actuation from off to on based on measured data.
Override timeout	Hardcoded for testing purposes.	Default configuration value from lighting control system.
Dead band for scheduled changes in actuation state	Hardcoded for testing purposes.	A dead band to prevent a false positive due to small differences in sensor report times around schedule points. Something that would otherwise be marked a fault is allowed if it is within the dead band and as long as the system is correct by the time it exits the dead band.

The diagnostic agent revealed the presence of faults in both zones upon initial configuration of the schedule-based controls, during the month of June 2014. Weekends were not analyzed, because there are not generally occupants present on weekends, and occupant switch use is the basis of Fault 3 (appropriateness of scheduled on/off hours) and Fault 4 (appropriateness of the length of the after-hours override timeout). Tables 4 and 5 summarize the presence or absence of faults for each of 4 weeks of analysis, for Zone 1 and for Zone 2. These results are also shown in the results dashboard that is included in the Transactive Lighting project website [LBNL 2014b]. A graphical representation of these faults is provided in Figures 13 and 14, including the diagnostic output from the agent, which summarizes the causes of each fault instance.

Table 4: Presence and absence of faults detected by the diagnostic agent in Zone 1

	Fault 1 actuation according to *programmed* schedule	Fault 2 actuation according to manager's *intended* schedule	Fault 3 potential to improve efficiency of occupant experience by modifying scheduled on/off times	Fault 4 potential to improve efficiency or occupant experience by modifying scheduled on/off times
Work Week 1, Jun 2-6	**Y**	Validated during the base case	N	N
Work Week 2, Jun 9-13	**N**		N	N
Work Week 3, Jun 16-20	**Y**		N	N
Work Week 4, Jun 23-27	**Y**		N	**Y**

Table 5: Presence and absence of faults detected by the diagnostic agent in Zone 2

	Fault 1 actuation does not match *programmed* schedule	Fault 2 actuation does not match manager's *intended* schedule	Fault 3 potential to improve efficiency of occupant experience by modifying scheduled on/off times	Fault 4 potential to improve efficiency or occupant experience by modifying scheduled on/off times
Work Week 1, Jun 2-6	**Y**	Validated during the base case	N	N
Work Week 2, Jun 9-13	N		N	N
Work Week 3, Jun 16-20	**Y**		N	N
Work Week 4, Jun 23-27	**Y**		**Y**	**Y**

June 2014 - Blue Zone							
SUN	MON	TUE	WED	THU	FRI	SAT	Faults Detected
1	2	3	4	5	6	7	Failure to sweep lights on/off as scheduled.
8	9	10	11	12	13	14	
15	16	17	18	19	20	21	Failure to sweep lights off as scheduled.
22	23	24	25	26	27	28	Potential to improve efficiency by reducing the length of the override time-out; incorrect after-hours switch response.
29	30						Failure to sweep lights on as scheduled.

Summary

2014-06-02: Fault detected: Failure in comms or actuator.
First seen: 18:15:20
2014-06-03: Fault detected: Failure in comms or actuator.
First seen: 08:15:19
2014-06-18: Fault detected: Failure in comms or actuator.
First seen: 18:15:35
2014-06-19: Possible fault: Potential failures - cmd log, comms, or actuator.
First seen: 20:23:50
2014-06-24: Fault detected: Failure in comms or actuator.
First seen: 19:47:19
2014-06-27: Fault detected: Failure in comms or actuator.
First seen: 18:15:19
2014-06-30: Fault detected: Failure in comms or actuator.
First seen: 19:08:20

Figure 13. Graphical representation of faults detected and diagnosed by the diagnostic agent, for Zone 1

June 2014 - Gold Zone							
SUN	MON	TUE	WED	THU	FRI	SAT	Faults Detected
1	2	3	4	5	6	7	Failure to sweep lights on/off as scheduled.
8	9	10	11	12	13	14	
15	16	17	18	19	20	21	Failure to sweep lights off as scheduled.
22	23	24	25	26	27	28	Potential to improve efficiency by extending scheduled hours-on; potential to improve efficiency by reducing the length of the override time-out; incorrect after-hours switch response.
29	30						Failure to sweep lights on as scheduled.

Summary

2014-06-02: Fault detected: Failure in comms or actuator.
First seen: 18:15:20
2014-06-03: Fault detected: Failure in comms or actuator.
First seen: 06:15:19
2014-06-18: Fault detected: Failure in comms or actuator.
First seen: 19:39:35
2014-06-19: Possible fault: Potential failures - cmd log, comms, or actuator.
First seen: 20:23:50
2014-06-27: Fault detected: Failure in comms or actuator.
First Seen: 20:34:19
2014-06-30: Fault detected: Failure in comms or actuator.
First seen: 06:15:19

Figure 14. Graphical representation of faults detected and diagnosed by the diagnostic agent, for Zone 2

The preceding plots and tables summarize the presence and absence of faults in each control zone. The specific nature of these faults, by date are as follows:

June 2nd, no sweep to 'off' according to schedule
June 3rd, no sweep to 'on' according to schedule
June 18th and June 19th, status of lights after-hours does not match schedule (switch data is insufficient to determine the cause)
June 24th, switch does not correctly actuate lights after-hours
June 27th, switch does not correctly actuate lights after-hours, and lights fail to turn off after elapse of the time-out
June 30th, no sweep to 'on' according to schedule

Figures 15-17 contain snapshots of data to illustrate that the outputs of the diagnostic algorithms are sound, based on the measured data from the office test space.

Figure 15 shows examples from two days in Zone 1 (June 2nd and June 3rd), in which the lights were not swept 'off' or 'on', according to schedule; on June 4th, the system was actuated correctly. In this plot, the relay time series data for the zone shows that the diagnostic algorithm correctly identified the presence of Fault 1.

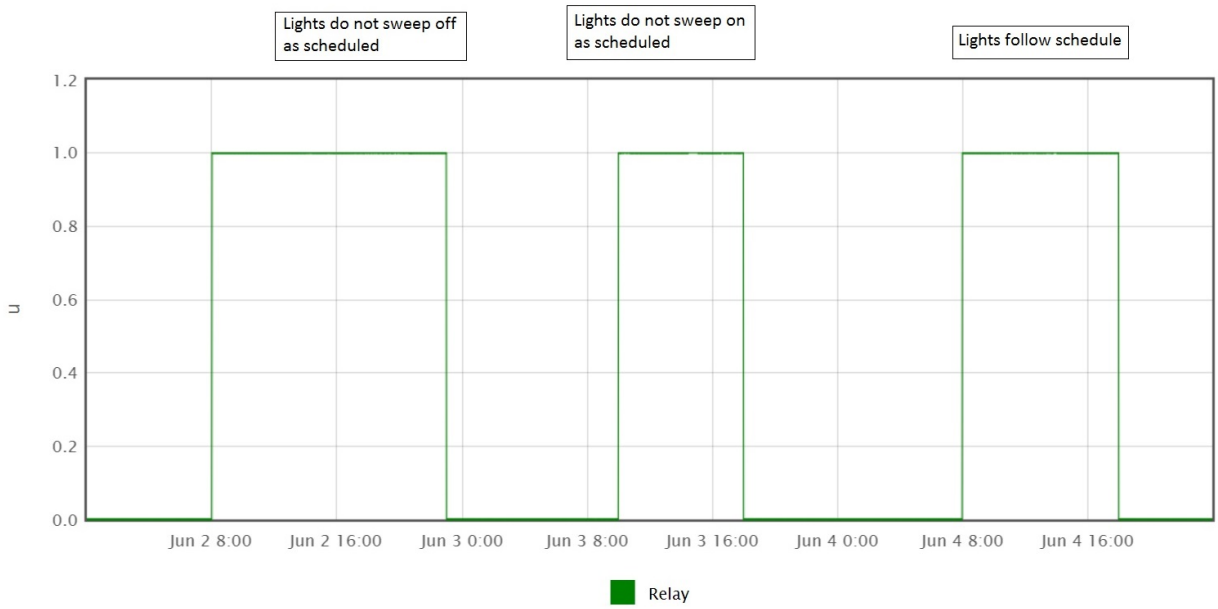

Figure 15. Data snapshot from two days for which the lights did turn off or come on according to schedule.

Figure 16 shows two of the days from Zone 1, in which the occupants regularly overrode the scheduled off event, to 'buy' more time, shortly after the scheduled off time. In the left-hand portion of the upper image (June 24th), the lights are swept off at 6pm, followed by an occupant override to 'on' at 6:05; in the lower image (June 27th), the lights are again swept off at 6pm, followed by an occupant override at 6:30pm. This reflects the potential to improve the occupant experience by modifying the schedule so that the sweep to off occurs later in the evening. Were this fault to appear regularly over a period of many weeks, the building manager might chose to modify the schedule. Note that in both cases, there is also occupant presence and manual switching later in the evening, however these events are much further away from the scheduled off time, and outside of the window of time considered by the diagnostic agent.

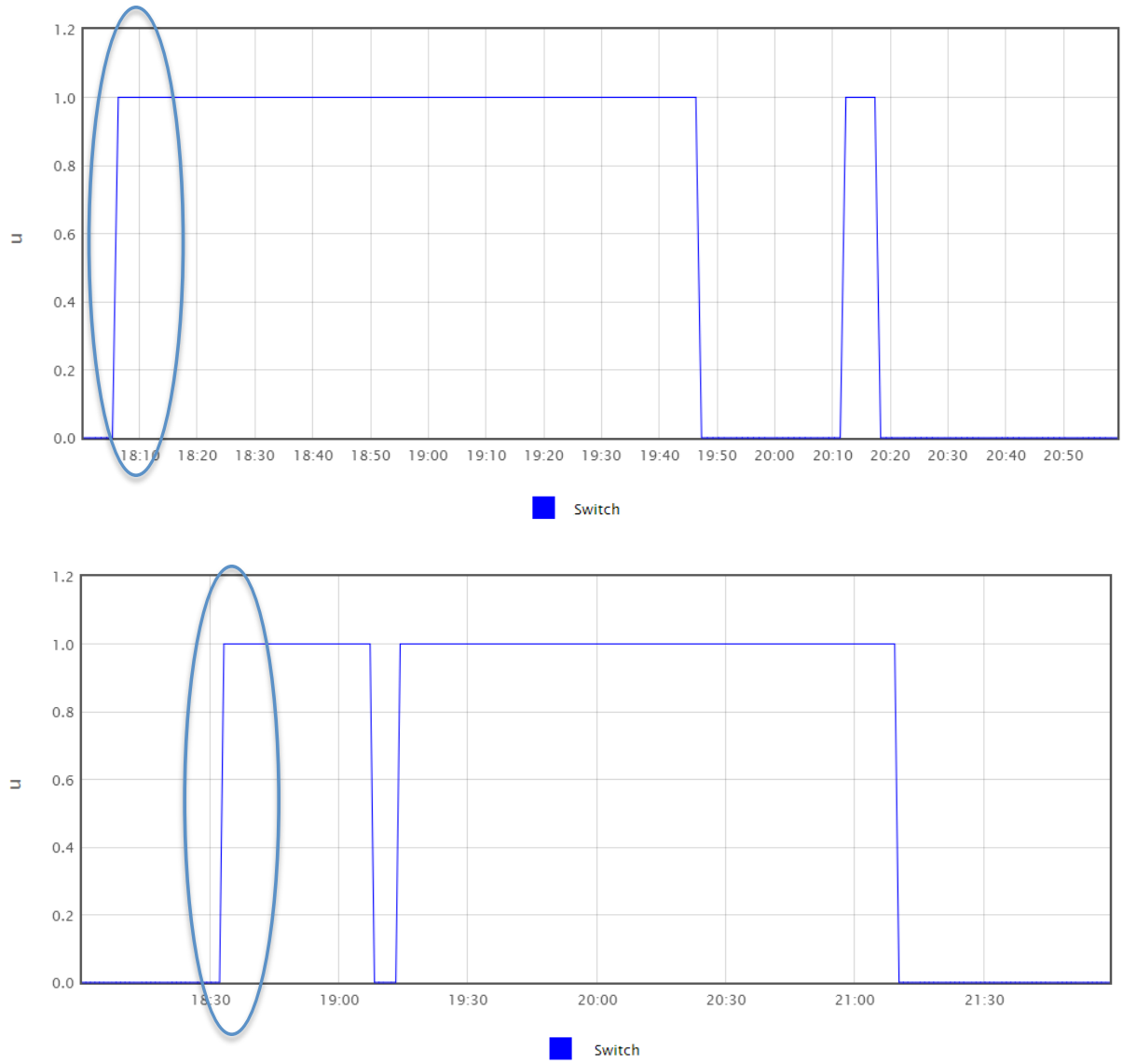

Figure 16. Data snapshots of two instances when the scheduled off-time occurred before vacancy in Zone 1

Figure 17 corresponds to cases in which the occupants turned the lights off before the elapse of the 2-hour override timeout. This reflects the potential to improve efficiency by reducing the duration of the timeout. Were this fault to appear regularly over a period of many weeks, the building manager might chose to modify the length of additional time that the control system provides during after-hours overrides to 'on'. For example, note that in the first instance on the left, the switch to "off" is immediately followed by a switch back to "on", which could indicate an unintended occupant switch event.

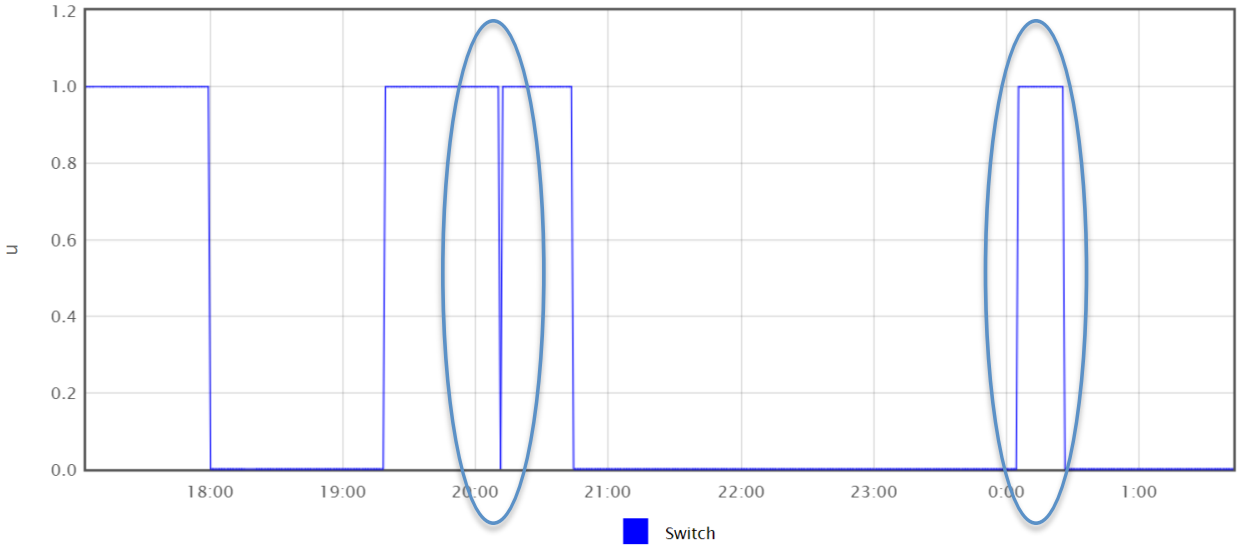

Figure 17. Data snapshot showing after-hours manual switches to off, before elapse of the override timeout.

Following identification of the presence of scheduling faults, the control system settings were reconfigured, and the diagnostic agent was used to confirm correct system behavior. Figure 18 contains a data snapshot for a week of operation for Zone 2, which shows regularly scheduled sweeps to 'on' at 6am, and sweeps to 'off' at 6pm, indicated by both the zone relay and power trends. On the first day of data, there is an after-hours manual override by the user, resulting in an additional two hours of lighting.

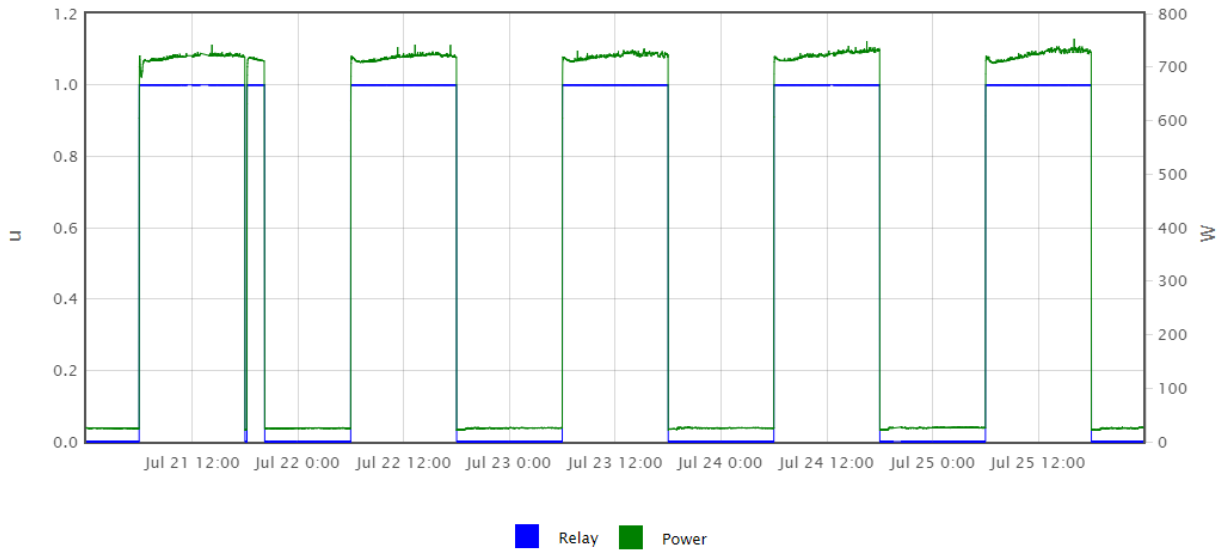

Figure 18. Data snapshot showing fault-free scheduled operations in Zone 2.

5.3.1 *Additional Fault Detection and Diagnostic Results*

In addition to implementation in the Volttron testbed described in Section 4, the diagnostic algorithms were also applied to system-level lighting data from three buildings in which scheduling controls were previously implemented. The results from these additional building cases are presented here. Table 6 summarizes the characteristics of these three buildings, the spaces in which scheduled was control was present, and the associated control systems.

Table 6. Characteristics of additional sites used to validate the performance of the FDD logic

Building Type	Site ID	Space Types	Control System
Office	Site 1	Exterior, conference rooms, corridors, office spaces	Lighting automation panels with GUI software, web-accessible PC
Office, classroom, laboratory	Site 2	Classroom, office, laboratory, corridors, exterior	Web-accessible BAS-integrated lighting automation panels
	Site 3	Classroom, laboratory, office	Lighting automation panels with GUI software, PC not web-accessible

5.3.2 *Additional Validation of Fault 1, Lights Don't Turn On/Off According to Programmed Schedule*

At Site 2, over a 4-month period of test data, the FDD logic identified 113 occurrences of Fault 1 across 19 schedule zones. This translates to an aggregate fault-rate of approximately 5%. Figure 19 shows a histogram of the frequency of Fault 1 across each of the 19 zones. These faults were not limited to any single type of system error. One notable finding from the algorithm, reflected in Figure 19 was the extreme number of faults occurring on one day during the first half of the monitoring period. The building manager was able to use this information to identify a series of short power outages that corresponded to lighting outages.

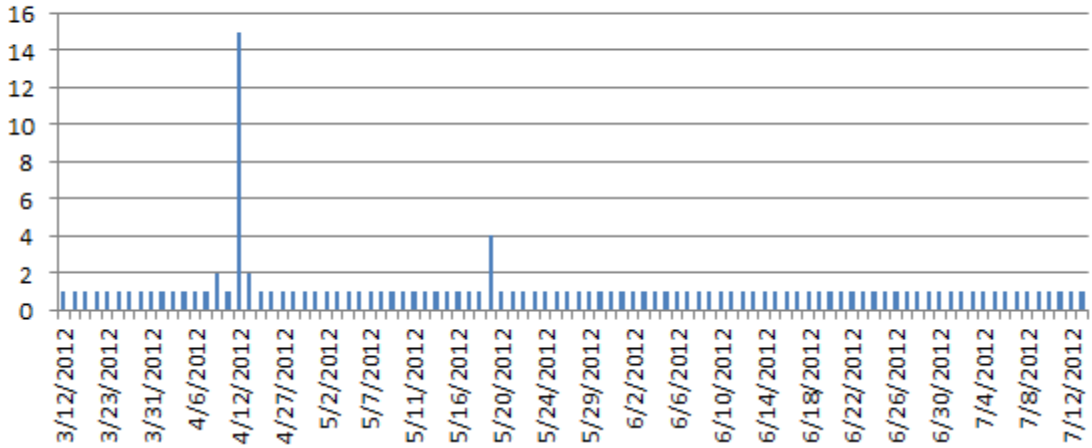

Figure 19. Histogram of the occurrence of Fault 1 across 19 scheduled zones at Site 2, for a 4-month period

Figure 20 shows a similar histogram, but for Site 3. In this building, there were 235 occurrences of Fault 1, across 55 relays, over approximately three months, for an approximate failure rate of 5%. Again, the chart shows a constant number of regular faults as well as a "spike" in faulting activity across multiple zones.

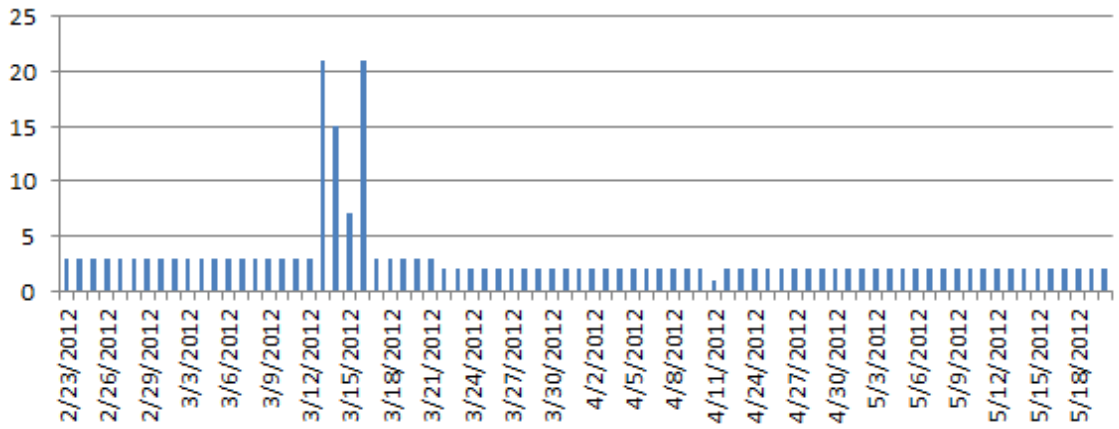

Figure 20. Histogram of the occurrence of Fault 1 across 55 relays at site 3, for a 4-month period

Figure 21 shows the same fault data as Figures 2 and 3, but by location/lighting relay rather than date. These charts clearly show "problematic" locations and relays in the buildings.

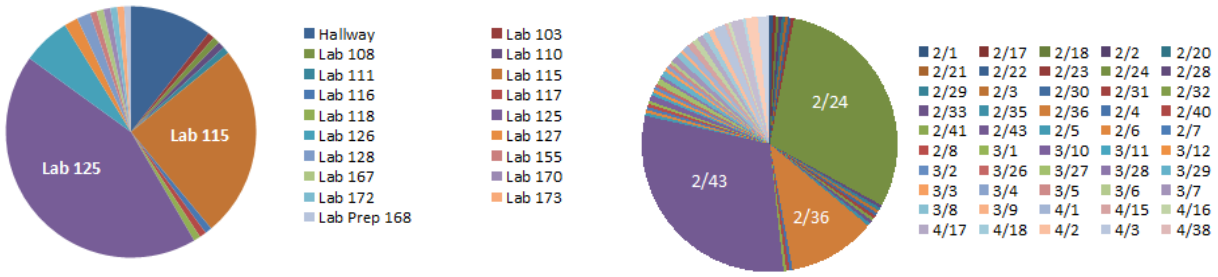

Figure 21. Occurrence of Fault 1 by location, over a four-month period at site 2 (left), and over a the three-month period Site 3 (right)

5.3.3 Additional Validation of Fault 2, Lights Don't Turn On/Off According to Manager's Intended Schedule

Site 1 was small enough that a record of the manager's intended schedule existed separate from, and external to the programmed controls. In that case, the manager was considerably less engaged with the control system GUI - logged system data from the field devices had not been pulled from the field servers since the system was installed (2 years prior). For that site, the manager retained a 'hard copy' paper-based schedule specification that was expected to reflect the schedules that were programmed into the control system.

At this site, the FDD algorithm showed that four of twenty-one functional control relays were not in fact being actuated according to the specified schedule; this mismatch between the actual programmed schedules and the specified schedules is shown in Figure 22. Throughout the 5 months of proof-of-concept demonstration data that were analyzed, these four relays, or control zones were logged as "on by schedule" one hour later than the manager had specified. These data prompted an investigation that revealed that the building had an alternate, supervisory BACnet interface to a separate lighting controller that had overridden the manager's intended programming.

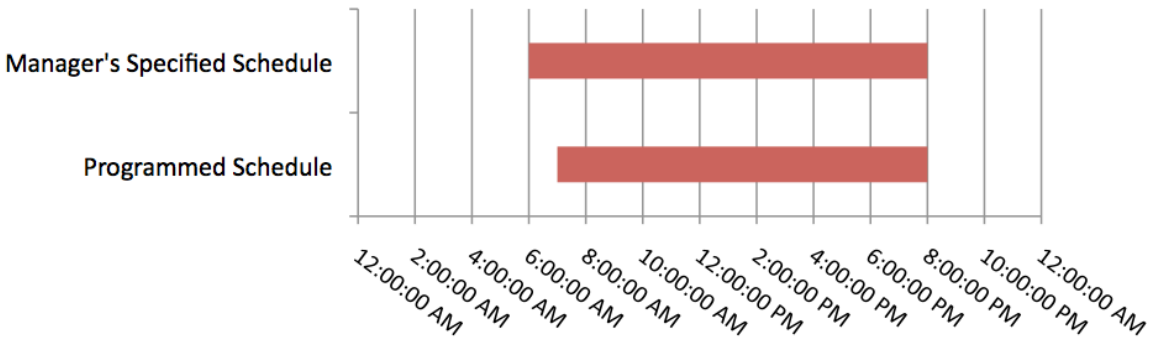

Figure 22. Chart showing the difference between the specified and programmed schedules at Site 1

5.3.4 *Additional Validation of Fault 3, Schedules Not Optimally Defined*

The logic to detect whether schedules are optimally defined is based on the frequency (percentage of days) of switch events to 'on' or 'off', soon after or prior to scheduled 'on' and 'off' times, as described in Section 3.4. For the purposes of testing the FDD logic, the following thresholds were applied:

Frequently = greater than 50% of the days in the analysis period
After scheduled off time = 1 to 4 hours
Prior to scheduled on/off time = 1 to 2 hours before

According to this structure, detection of the fault necessarily depends on the specific threshold values that were assigned. Due to the percentage-of-days parameter associated with the term 'frequently', detection also depended on the total number of days that are contained in the analysis period. Table 7 illustrates the impact of the length of the analysis period on the number of faults that were detected, when the algorithm was applied to the data from demonstration Site 2. These results indicate that periodic re-evaluation of schedules may be beneficial, if occupancy patterns fluctuate over time.

At Site 1 no occupant switch events were recorded during data collection period, so the occurrence of Fault 3 could not be analyzed.

Table 7: Impact of the length of the analysis period on the number of occurrences of Fault 3 (Site 2)

Length of time analyzed	# of zones exhibiting Fault 3 (out of 19 zones)
1 month	4
2 months	2
4 months	0

5.3.5 Additional Validation of Fault 4, Override Time-outs Not Optimally Defined

Similar to Fault 3, the logic to detect whether the override time-outs are optimally defined is based on the frequency of switch events to off or on, soon after or prior to elapse of the time-out period, as described in Section 3.4. For the purposes of testing the FDD algorithm, the following thresholds were applied:

Frequently = greater than 50% of the instances occurring in the analysis period
Soon after elapse of the timeout = within 5 minutes
Prior to elapse of the timeout = 30 minutes or greater

According to this structure, as with Fault 3, detection of Fault 4 necessarily depended on the specific threshold values that were assigned, and the duration of the analysis period.

At Site 1 no occupant switch events were recorded during data collection period, so the occurrence of Fault 4 could not be analyzed. However, Fault 4 was detected at Sites 2 and 3. Table 8 contains the aggregated occurrence of Fault 4 across all zones in both of the buildings. The first four rows in the table correspond potential to reduce the length of the time-out, increasing efficiency, whereas the last four rows correspond to the potential to increase the time-out length to improve service levels.

In both buildings there was potential to reduce the length of the time-out to increase efficiency, and reflecting consistency of findings, in neither case did the data indicate potential to improve service levels by increasing the length of the time-out. At site 3, the observed frequencies of 52% and 43% are quite close to the 50% threshold, providing a good illustration of the dependency of the detection result on the selected threshold value.

Table 8: Occurrence of Fault 4, suboptimal override time-outs, in Site 2 and Site 3

	Site 2	Site 3
Total number of after-hours time-out cases	756	4232
# manual-off events prior to (30 min) elapse of time-out	552	2221
Rate	**73%**	**52%**
Potential to reduce time-out length to increase efficiency?	**Yes**	**Yes**
# of after hours time-out cases	149	518
# of manual-on events 'soon after' (within 5 min of) elapsed time-outs	55	224
Rate	**37%**	**43%**
Potential to increase time-out length to increase comfort?	**No**	**No**

6 Conclusions and Future Work

This project demonstrated the concept of measurement and verification (M&V), and fault detection and diagnostic (FDD) agents for scheduled lighting controls utilizing the Department of Energy's Transactive Network platform, to include applications for the lighting end use, complementing and expanding prior work that focused on rooftop HVAC end uses. This project is important because it focuses on the concepts of M&V and FDD by leveraging DOE's Transactive Network platform and applying it to lighting controls. Together, lighting and HVAC comprise the majority of energy use in commercial buildings. Moreover, this project highlights the need for all DOE projects to incorporate these concepts in the development of Transactive Network agents, and DOE's need to develop a spectrum of baselines and metrics for buildings against the different user groups and use cases of all transaction based services.

All DOE -funded agents will be made available to the public through open source licenses. These agents can be integrated with new and existing user-interface diagnostic and analytical software, web-based visualization applications, and ideally, into the operator-facing software packages that are provided by third party control companies or other interested parties.

By providing both diagnostic and M&V capability for the Transactive Network platform, two challenges associated with the adoption and effective use of controls can be overcome: 1) lack of feedback into ongoing energy savings that the controls deliver; 2) identification and prevention of configuration and operational faults that can negatively impact occupant experience or comfort, and achieved energy savings.

Specific to the use of scheduling control in existing buildings, three additional challenges and opportunities are addressed in this work: 1) although scheduling is one of the simpler lighting

control strategies, it is implemented correctly and on an ongoing basis in buildings, resulting in energy waste, or poor service or comfort levels for building occupants; 2) building managers, do not commonly have the time or ability to manually generate and resolve trend log reports and plots to identify improvement opportunities; 3) automated fault detection and diagnostic (FDD) routines that 'push' recommendations to building managers may streamline the process of identifying energy waste, and control problems, enabling improved operations.

Diverse control strategies can be used to efficiently control the lighting in commercial buildings, including scheduling, occupancy sensing, setpoint tuning, bi- or multi-level switching, and daylight-responsive dimming. The completed work began with a focus on scheduled lighting controls to demonstrate one of the strategies on the simpler end of the spectrum, that does not require direct-ballast control, yet is still under adopted in the commercial building sector. Follow-on work may address occupancy-based controls, and more sophisticated control strategies that often entail higher capital investment and offer more advanced control logic and sensing capabilities.

It is important to emphasize that in this work, the objective was to demonstrate the automated measurement and verification concepts that can, on an ongoing basis, quantify *project specific* savings that are realized by implementing lighting control in a given space. This is in contrast to demonstration studies that have focused on establishing general savings levels that can be expected from one control strategy versus another. For example, it is generally accepted that scheduling can save five to fifteen percent in energy use. However, given the atypical occupant behavior in the demonstration site, the measured and verified savings were on the order of sixty percent. These savings are significantly higher than those generally expected when implementing schedule-based controls. This was due to the fact that energy use in the base case was extremely high because of severely underused manual controls; occupants commonly left the lights on all night and all weekend in both demonstration zones.

The savings measurement and verification approach can be augmented to address more complex base cases, or a variety of mathematical baselining approaces. In this work, the focus was to develop automated approaches to quantifying savings, for one of the simplest types of lighting control (zone-level scheduling), relative to the most common case in today's commercial buildings (no controls automation). Since occupant-controlled lighting is a non temperature-dependent load, and since the control was zone-based, a simple model was sufficient. Energy use could be robustly characterized by variables that represent: a) occupied vs. unoccupied blocks of time, and b) occupied vs. unoccupied days of the week.

However, for base cases that do not represent simple energy use profiles, more sophisticated models should be developed and applied. For example, consider a case in which the energy savings due to an *improved* system for occupancy- and photo-controlled automated lighting is implemented (i.e., both the base case and post case include occupancy detection and daylight dimming). The model for the base case would require the addition of variables to capture the effects of occupancy as well as natural daylight levels, and time dependencies more resolved than day of week and occupied vs. unoccupied blocks of time. The same is true of the baseline

models used to characterize upgrades to the control of weather-dependent loads, such as HVAC; they require the addition of outside air temperature and perhaps humidity, and also more resolved representations of time.

A promising area for future research is to begin to understand the minimal number of models and explanatory variables that are needed to quantify energy savings across the diversity of end uses and control strategies that are commonly implemented to improve the operational efficiency of commercial buildings. A *new model* for every situation is *not desirable*, however a *single model* cannot be used for all situations.

The lighting M&V agent that was presented in this work is one example of savings estimation for a specific type of transactional interaction – "end-user services", which includes operations and diagnostics of end-use assets [Soumasundaram 2014]. In considering the full set of transactional interactions, which extends to energy market services, grid services, and societal services, a much broader spectrum of M&V and financial settlement approaches must be developed for the diversity of use cases that those interactions entail. For example, Somasundaram et al. [2014] describe a use case in which tenants might receive a monthly energy allowance, against which rebates, penalties, and exchanges with other tenants could be assessed. The continuum of measurement, verification, and settlement approaches a use case of this type would look quite different from those associated with long-term verification of the savings from and end-use specific control strategy (as for lighting case presented in this work).

The measurement and verification of services operating in a market (structured or unstructured) requires settlement methods. These methods must operate across the domains of all available transaction-based controls and may push and pull the development of new services solutions – for example, comfort-based services. Uncovering how to measure and verify these service delivery opportunities may even yield new solutions and savings left unconsidered or undiscovered before the measurement metric was determined. Therefore, careful consideration must be made to the measurement metric and the owner of the use and business cases that the metric serves. Accordingly, this continuum of requirements and approaches for M&V to support the diversity of transaction-based services and use cases is a larger topic that will be explored in future work.

References

Haack, JN, Katipamula, S, Akyol, BA, Lutes, RG. October 2013, "VOLTTRON Lite: Integration Platform for the Transactional Network." Pacific Northwest National Laboratory. PNNL-22935.

Katipamula, S, Lutes, RG, Ngo, H, Underhill, RM. "Transactional Network Platform: Applications." October 2013, Pacific Northwest National Laboratory. PNNL-22941.

Lawrence Berkeley National Laboratory (LBNL) [Internet]. "FLEXLAB™. What We Offer|FLEXLAB." Environmental Energy Technologies Division, 2014. Available from http://flexlab.lbl.gov/what-we-offer; accessed September 30, 2014.

Lawrence Berkeley National Laboratory (LBNL) [Internet]. "Transactional Network." Environmental Energy Technologies Division, 2014. Available from http://transactionalnetwork.lbl.gov; accessed September 30, 2014.

Navigant Consulting, Inc. 2010 "US Lighting Market Characterization. Report prepared for Solid State Lighting Program, Building Technology Program," Energy Efficiency and Renewable Energy, US Department of Energy. July 2012.

Oak Ridge National Laboratory (ORNL) [Internet]. "Transactional Network Platform (TNP)." Oakridge National Laboratory, 2014. Available from http://btric.ornl.gov/tnp.shtml; accessed November 6, 2014.

Piette, MA, Brown, R, Price, P, Page, J, Granderson, J, Reiss, D, et al. "Automated measurement and signaling systems for the transactional network." December 2013, Lawrence Berkeley National Laboratory. LBNL-6611E.

Somasundaram, S, Pratt, RG, Akyol BA, Fernandez, N, Foster, N, Katipamula, S, et al. "Reference guide for a transaction-based building controls framework." April, 2014, Pacific Northwest National Laboratory. PNNL-23302.

UC Berkeley [Internet]. "sMAP: the Simple Measurement and Actuation Profile – sMAP 2.0 Documentation." University of California Regents, 2013. Available from http://www.cs.berkeley.edu/~stevedh/smap2/index.html, accessed April 30, 2014.

WattStopper [Internet]. "Digital Lighting Management." 2014. Available from http://www.wattstopper.com/products/digital-lighting-management.aspx#.U1fXnV5cJuI; accessed April 23, 2014.

Williams, A, Atkinson, B, Garbesi, K, Rubinstein, F, Page, E. "A meta-analysis of energy savings from lighting controls in commercial buildings." September, 2011, Lawrence Berkeley National Laboratory. LBNL-5095E.

Appendices

This appendix contains pseudocode that describes how the logic in the fault detection and diagnostic, measurement and verification agents is implemented in code. The logic for each of the agents is described in Sections 3.2-3.3 in the main body of this report, and the code for the agents are available for download from https://bitbucket.org/berkeleylab/eetd-tn-lighting. The agents were developed in the python programming language – the pseudocde is provided to enable developers to implement these agents in a programming language of their choice. This appendix also contains code-level documentation for developers who wish to implement the agents presented in this report into their own projects. This code-level documentation is also included in the distribution source code distribution.

For ease in interpretation, the main body of the report refers to the 'agents' as if they were two single sets of code. However in reality, the FDD and M&V applications consist of multiple parts: 1) a means of interfacing with the Volttron message bus, and 2) diagnostic logic and logic to calculate a baseline and quantify energy savings. To be more accurate, one might label only the portion of code that interfaces with the Volttron message bus as the 'agent', and the code that contains the FDD and savings calculations as application 'modules'. These concepts are illustrated in Figure A-1 below, and introduced to add clarity to the pseudocode and code-level documentation that follow.

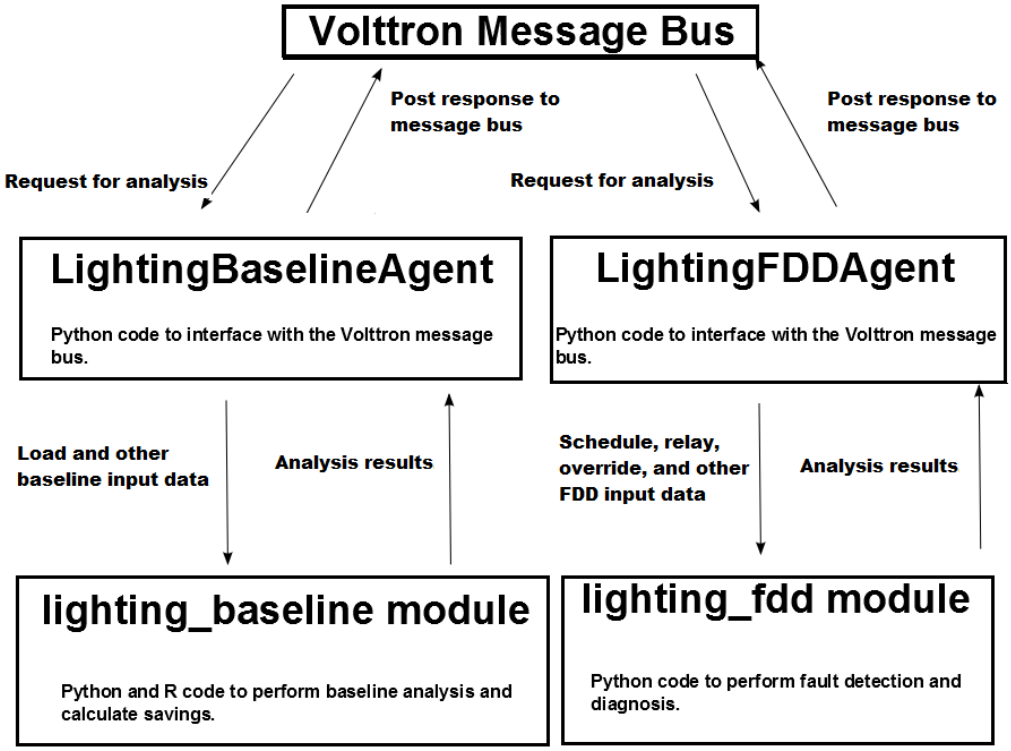

Figure 23. Schematic Diagram of Diagnostic and M&V Code

Appendex A.

Psuedocode for the Measurement and Verification Agent

Interfacing With the Volttron Message Bus

1. Subscribe to the Volttron topics "lighting_analysis/summary_request" and "lighting_analysis/comparison_request"
    a. NOTE: Since these two topics share quite a bit only the differences in processing the two requests will be noted below.
2. Wait for a message to be posted to either topic
3. Extract relevant fields from the header and message of the request
    a. From the header try to get the following fields:
        i. "requestorID"
        ii. "From"
    b. From the message try to extract the following and create a dictionary of their values:
        i. "timezone"
        ii. "sq_ft"
        iii. "workdays"
        iv. "workday_start"
    c. For summary_request messages also extract:
        i. "load_data"
    d. For comparison_request messages also extract:
        i. "baseline_load_data"
        ii. "comparison_load_data"
4. Create a response header by using the request header
    a. Copy the request header.
    b. Changing the "To" field value to the "From" field value from step 2a.
        i. If there was no "From" value use "None" instead
    c. Set the "From" value to "LightingBaselineAgent".
5. Create a relevant analysis object
    a. For summary_request create a Lighting_Analysis object.
        i. Use the dictionary created in step 2 as keyword arguments in the constructor.
    b. For comparison_request create a Lighting_Compare object.
        i. Use the dictionary created in step 2 as keyword arguments in the constructor.
6. Perform the analysis
    a. For summary_request if there was no load_data in step 3.c return a dictionary with an "error" key and message saying it is required as a value.
    b. For comparison_request if there was no baseline_load_data or no comparison_load_data in step 3.d return a dictionary with an error key and message saying both are required as a value.
    c. For summary_request call Lighting_Analysis.summary with the "workday" and "workday_start" values from step 3.b as keyword arguments

    d. For comparison_request call Lighting_Compare.compare with the “workday” and “workday_start” values from step 3.b as keyword arguments
7. Create the response message contents
    a. For summary_request create a dictionary with one key: “summary_stats” with the value the result from step 6.
    b. For comparison_request create a dictionary with one key: “comparison_stats” with the value the result from step 6.
8. Publish response
    a. Create response topic.
        i. For lighting_analysis the response topic is “lighting_analysis/summary_response/{requestor_id}” where {requestor_id} is the value from 3.a.i. If that value is not present use “None” instead.
        ii. For lighting_summary the response topic is “lighting_analysis/comparison_response/{requestor_id}” where {requestor_id} is the value from 3.a.i. If that value is not present use “None” instead.
    b. Call publish_json with the response topic from 8.a as the first parameter, the headers from step 4 as the second parameter and the dictionary from step 7 as the third parameter
9. Go to step 2

Calculating a Baseline and Quantifying Savings:

1. Read command-line arguments:
    - loadFile : name of file containing timestamp, load data (kW).
        - if filename not specified, terminate with error.
    - holidayFile : name of file containing dates of holidays.
        - if filename not specified, variable takes null value.
    - summaryStatisticsFile : name of file for output.
    - workdays : string such as “12345” specifying which days of week are workdays.
        - Days of week run from 0=Sunday through 6=Saturday.
        - Default = “12345”.
    - workdayStart : time of start of 12-hour workday.
        - Specified in HH:MM.
        - Default = “06:30”.
    - verbosity : determines amount of progress information to be sent to standard output during running of the program. Used for debugging.

2. Read input files
    - Read load data from file specified by loadFile.
    - Read holiday dates from file specified by holidayFile [This has not yet been implemented as of 9/2014].

3. Aggregate the load data to 15-minute intervals.

- From load data, calculate E(timestamp) = cumulative energy used from start of data to each reported timestamp.
- Find t0 = first output interval that is at least 15 minutes after the start of the data.
- For each 15 minutes beginning at t0, i.e. for t_n = t0 + 15*n where n is an integer 0 or higher, interpolate E(timestamp) to find E(t_n).
- Calculate e(t_n) = energy used during the interval = E(t_n) – E(t_n-1)
    - If the load data have timestamps in the interval, but are missing data associated with those timestamps, then count what fraction of data from the interval are missing.
        - If more than 50 percent of data from the interval are missing, set e(t_n) to NA
        - If fewer than 50 percent of data from the interval are missing, then set e(t_n) = e(t_n)/(1 - fraction of data that are missing).
- Convert energy in the 15-minute interval to power (kWh) by dividing by 0.25 hours.

4. Calculate summary statistics
    - Parse the workdayStart and workdays strings, and determine the days of the week that are workdays, and the hours of those days that are work hours.
    - Calculate mean power for each non-holiday workday during work hours, ignoring any time periods that have no data.
    - Calculate mean power for each non-holiday workday during non-work hours, ignoring any time periods that have no data.
    - Calculate mean power for each non-holiday non-workday during work hours, ignoring any time periods that have no data.
    - Calculate mean power for each holiday during work hours, ignoring any time periods that have no data.
    - Calculate mean power for each holiday during non-work hours, ignoring any time periods that have no data.
    - Write all of the mean power values to the file specified by variable summaryStatisticsFile

5. Terminate program

Appendex B.

Psuedocode for the Fault Detection and Diagnostic Agent

Interfacing With the Volttron Message Bus

1. Subscribe to the Volttron topic “lighting_fdd/request”
2. Wait for a message to be posted to the “lighting_fdd/request” topic
3. Extract relevant fields from the header and message of the request
    a. From the header try to get the following fields:
        i. “requestorID”

        ii. “From”
    b. From the message try to extract the following and create a dictionary of their values (if they are present in the message):
        i. “implemented_schedule”
        ii. “relay_timeseries”
        iii. “override_timeseries”
        iv. “intended_schedule”
        v. “load_timeseries”
        vi. “expected_load_min_change”
        vii. “expected_load_max_change”
        viii. “relay_time_comparison_epsilon”
        ix. “override_timeout”
        x. “compress_override_to_change_of_state”

4. Create a response header by using the request header
    a. Copy the request header.
    b. Changing the “To” field value to the “From” field value from step 2a.
        i. If there was no “From” value use “None” instead
    c. Set the “From” value to “LightingFDDAgent”.
5. Create a Lighting_FDD object
    a. Check if “implemented_schedule”, “load_timeseries”, and “override_timeseries” were all present in step 3.b
        i. If any are missing return a dictionary with an “error” key and a message saying what is missing. Go to step 7
    b. Pass the dictionary created in step 2b as keyword arguments to the Lighting_FDD constructor: Lighting_FDD(**dict_from_step_2_b)
6. Run the analysis
    a. Call the Lighting_FDD object from step 5’s get_faults() method.
    b. Store the results in 4 lists:
        i. faults_with_implemented_schedule
        ii. faults_with_intended_schedule
        iii. suggested_schedule_changes
        iv. suggested_override_timeout_changes
7. Create the response message contents
    a. If coming here from clause 5.a.i just use the returned value as the response message
    b. Else if step 6 completed create a dictionary from the results of step 6.
        i. The keys of the dictionary should be the string names in step 6.b

ii. The values should be the string values of the actual results from step 6 if there are any, else None

8. Publish response
    a. Create the response topic. It is "lighting_fdd/responses/{requestor_id}" where {requestor_id} is the value from 3.a.i. If that value is not present use "None" instead.
    b. Call publish_json with the response topic from 8.a as the first parameter, the headers from step 4 as the second parameter and the dictionary from step 7 as the third parameter.
9. Go to step 2

Detecting and Diagnosing Scheduling Faults

Required inputs:

- lighting schedule
- relay timeseries
- override timeseries
- relay time comparison epsilon (default 5 minutes)
- override timeout (default 2 hours)

Optional inputs. Indented inputs can provide context on their parent.

- load timeseries
    - minimum expected load change
    - maxumum expected load change
- time zone
- intended schedule
- system time vs real time timeseries

Explanation of inputs:

- lighting schedule:
  The expected state of the system at any given time.
- relay timeseries:
  A timeseries list of values indiciating whether the relay is on or off at a give time.
- override timeseries:
  A timeseries list of values indicating when override events happen.
- relay time comparison epsilon:
  The case where lights turn off exactly (to a binary comparison) at the scheduled time. This parameter defines a grace period around a schedule change that will not be flagged as a fault. This is especially needed if the time resolution on the relay timeseries is coarse.
- override timeout:
  The length of time the lights should remain on after a manual override before turning off.

- load timeseries:
  A timeseries list of values representing actual electrical load. This may be the load of the relay if this is not available a more aggragate load.
- minimum and maximum expected load change:
  If there is load data this is the range that the user expects it to change when the lights are actuated. Since the load stream may contain other devices the FDD will try to determin if the load changes by an expected amount when the lights are actuated rather than just check to see if the load goes to near zero.
- time zone:
  The time zone of the system if known
- intended schedule:
  There is potentially the case where the actual programmed system differes from the intended schedule. If there are seperate sources for each (e.g. reading it programmatically from a controller vs getting something from a document) the FDD can provide additional analysis.
- system time vs real time timeseries:
  Clocks can drift and this can affect diagnosis. If there is a record of this drift then it can also be provided to the FDD.

Fault checks:

Type 1:

1) Find all points in the relay timeseries where the reported status differs from the expected status as determined by the lighting schedule.
   a) If relay time comparison epsilon is provided exclude any points that are in the wrong state but within the epsilon of a time when the relay is scheduled to change state.
2) Take the list of points from Step 1 and divide them into two categories: those where there was a recent override and those where there was not. Recent is defined by the override timeout input value.
3) Step 3: For each point in without a recent override:
   a) If there is load data check if the load data changed as expected during the last scheduled change.
      i) YES: Report: No fault but failure in cmd log
      ii) NO: Report: Fault detected. Failure in comms or actuator.
   b) If there is no load data: Report: Possible fault, Potential failures - cmd log, comms, or actuator.
4) For each point with a recent override:
   a) If the override value is not the same value as the relay status point: Run Step 3 on the point and report the result.
   b) Else No Fault.
5) If system time vs real time timeseries is provided as an input report all instances of drift of the system time.

6) For the collection of points reported so far filter them to provide the start and end times for the fault. A fault starts on it's first occurrence and continues until A: there is a point that is not reported or B: there is another reported point with a different message.

Type 2:
1) Check the intended schedule against the required lighting schedule and report any cases where they conflict.
2) Run Type 1 again but with the intended schedule parameter instead of the required lighting schedule.

Type 3:
1) If the duration of scheduled off time is less than 8 hours for a weekday or 12 hours for a weekend: Report: Potential to improve efficiency by reducing scheduled hours-on.
2) Check if overrides happen close to scheduled points.
   a) If overrides to off occur frequently 1-2 hours before scheduled off times: Report: Potential to improve efficiency by reducing scheduled hours-on.
   b) If overrides to on occur frequently 1-2 hours before before scheduled on times: Report: Potential to improve service by extending scheduled hours-on.
   c) If overrides to on occur frequently 1-4 hours after scheduled off times: Report: Potential to improve service by extending scheduled hours-on.

Type 4:
1) Check if overrides happen close to the elapse of an override timeout.
   a) If overrides to off occur frequently at least 30 minutes prior to the elapse of the timeout: Report: Potential to improve efficiency by reducing the length of the override timeout.
   b) If overrides to on occur frequently and within 5 minutes of the elapse of the timeout: Report: Potential to improve service by increasing the length of the override timeout.

Appendex C.

Code-level Documentation

Purpose

This collection of applications provides methods for analyzing properties of lighting systems. The baseline applications provide a way to characterize lighting load and compare load behavior while the fault detection applications analyze the performance of automated lighting systems that operate on a schedule.
There is code provided to meet two basic use cases:

1. Import the basic calculation modules (lighting_baseline and lighting_fdd) into another application and use them as any other python module.
2. Run the provided agents (LightingBaselineAgent and LightingFDDAgent) in the Volttron environment. See https://github.com/VOLTTRON/volttron/wiki for more on Volttron.

System Requirements and Dependencies

All code was tested using Python 2.7 (2.7.5 specifically) on machines running the Ubuntu OS (12.04).

lighting_baseline module:

- R (available from http://www.r-project.org)
- The optparse R library.

lighting_fdd module:

- dateutil python library
- pytz python library
- tzlocal python library

Agents require the Volttron platform. Tested on 1.x versions. LightingBaselineAgent requires lighting_baseline and LightingFDDAgent requires lighting_fdd.

Additional Considerations and Concepts

Agents vs Modules: Modules perform analysis while agents handle the interface between modules and the Volttron message bus. Modules are able to be used in non-Volttron applications while Agents are fairly tightly coupled with the Volttron system.

Timeseries: Most of the data that these application use is in the form of a timeseries. A timeseries is a list of (timestamp, value) pairs where the unit of the value depends on the type of timeseries. For example, in a timeseries of daily temperatures the value might be in C or K or F. If the timeseries was of electrical current the values might be in W or kW. These applications are unit-agnostic. It is up to the caller to make sure they are passing data in units that agree.

Schedule: The fault diagnostic applications diagnose faults with respect to a desired schedule. This is the representation of when the lights are expected to be on and off. Diagnosis is made by first finding points where the system does not match the schedule and then attempting to provide context.

Installation Instructions

Install R and the optparse library. For a typical R installation, this can be achieved by launching R and, at the R prompt, typing:

```
install.packages("optparse")
```

Install Volttron. Eventually this should install everything else by default. However to install manually download the modules and agents from https://bitbucket.org/berkeleylab/eetd-tn-lighting and manually install each by navigating to each root directory and running “python setup.py install”.

Execution Instructions for using standalone modules

A. LightingSummary.R Module Execution Instructions

The program is run from the command line by giving a single command of the form

./LightingSummary.R –h

Or
./LightingSummary.R [-flag value]…

LightingSummary.R –h will produce the following output (the h stands for 'help'):

```
Usage: ./LightingSummary.R [options]


Options:
-l LOADFILE, --loadFile=LOADFILE
Name of load data file (Required)

-H HOLIDAYFILE, --HolidayFile=HOLIDAYFILE
Name of file containing holiday dates (Optional)

-s SUMMARYSTATISTICSFILE, --summaryStatisticsFile=SUMMARYSTATISTICSFILE
Name of output file for summary statistics [default summaryStatisticsFile.csv]

-w WORKDAYS, --workDays=WORKDAYS
Which days are workdays? 0 = Sunday, etc. [default 12345]

-W WORKDAYSTART, --workdayStart=WORKDAYSTART
When is start of 12-hour 'workday'? [default 6:30]

-v VERBOSITY, --verbosity=VERBOSITY
determine what progress and error reports to print (non-neg integer) [default 1]

-h, --help
Show this help message and exit
```

As the help output indicates, in addition to –h there are six case-dependent flags, of which only one, the load flag and its associated file name, is required. The order of the flags on the command line does not matter.

To invoke the program using defaults for all of the arguments, the command is
./LightingSummary.R –l *loadFileName*
where *loadFileName* is replaced by the name of the file that contains the load data. Input file formats are discussed below.

As the help output indicates, the default workdays are Monday-Friday but can be changed with the –w flag, and the default for "workday hours" is from 6:30 a.m. to 6:30 p.m. The "work day" is always 12 hours long but the start time can be changed by using the –W flag.

Optionally, a file of holiday dates may be specified using the –H flag.

For example, to define workdays as Tuesday through Saturday, and work hours as 8:00 a.m. to 8:00, p.m., with holiday dates provided in a file called Holidays.csv, and program output written to a file called summaryOutput.csv, the command is
./LightingSummary.R –l loadFileName –w 23456 –W 8:00 –H Holidays.csv -s summaryOutput.csv

B. Input File Formats

Load data file

The file containing load data is comma-delimited. Any lines that begin with # are ignored, wherever they occur in the file. All other lines must have at least two fields (columns) and must have the same number of columns. If more than two columns are present, only the first two are used. The first column contains a timestamp, the second contains the load measurement associated with that timestamp.

The timestamp may be in any of the following formats:
Seconds since 1970-01-01 00:00:00
Milliseconds since 1970-01-01 00:00:00
Year-month-day hours:minutes:seconds, e.g. 2014-08-28 16:30:00
The program automatically determines what time format is being used: if the first timestamp in the file includes a ":" then the program assumes format 3 is being used. If it does not, then numbers exceeding 3×10^9 are assumed to be in milliseconds, otherwise they are assumed to be in seconds. Timestamps do not need to be at any particular frequency and do not need to fall on round numbers of minutes or hours. They are assumed to be regularly spaced in time; if they are not, no error is generated but the program's results may be incorrect.

The second column contains load data, and must be an integer, real number, or the string NA (without quotes). Below are the first few lines of a legal input load data file:

```
# This is a comment line
2013-07-22 16:37:46, 7.976
2013-07-22 16:47:46, 7.424
2013-07-22 16:57:46, 7.700
2013-07-22 17:07:46, 7.412
2013-07-22 17:17:46, 7.544
```

The file may contain additional columns as well; those columns are ignored.

Holiday file

The optional holiday file is comma-delimited. Any lines that begin with # are ignored, wherever they occur in the file. All other lines must have the same number of commas (which may be zero). The first column, which may be the only column, contains a timestamp in the format year-month-day, for example 2014-07-04. Other columns, if they exist, are ignored.

The following are three lines of a legal holiday file:
2014-04-07, Independence Day
2014-09-01, Labor Day
2014-10-13, Columbus Day

C. Output
The output file is comma-delimited. Its default name is summaryStatisticsFile.csv but this may be changed as described in the Execution Instructions section. The file has two fields (columns) per line and has no header line. The first column specifies the time category being reported, the second specifies the mean load during that time category. Here is an example output file:
workdayDayMean,9.92
workdayNightMean,6.18
nonworkdayDayMean,3.3
nonworkdayNightMean,4.78
holidayDayMean,NaN
holidayNightMean,NaN
nonHolidayWeekMean,6.9

Units of the output are the same as units of the input. For example, if the load data are provided in kW then the output is in kW. If (as in this example) no holidays are specified then NaN is reported for the mean load during holiday days and nights.

Notes
As of September 2014 there is one known implementation issue: The 12-hour work "day" is assumed to end before midnight, and if this is not the case there may be no way to get the program to perform exactly the calculations that are desired.

For example, suppose working hours are 5 p.m. to 5 a.m. (that is, work takes place at night) on Wednesday through Saturday nights. A nightclub would be a relevant example. If one defines the start of the workday with the flag –W 17:00 and the workdays with the flag –w 3456, the result will not be as intended: When calculating workdayDayMean, the program will include all time periods that are (during Wednesday, Thursday, Friday, or Saturday) and are also (between 5pm and midnight or between midnight and 5 a.m.). By this logic, the early morning hours on Sunday will not be included even though they should be, and the early morning hours on Wednesday will be included even though they shouldn't be.

lighting_baseline Execution Instructions
The program is intended to be run as any other python module. Import lighting_baseline and construct either a Lighting_Analysis or a Lighting_Compare object based on the use case.
Use Lighting_Analysis.summary() to analyze one one monitoring period.
Use Lighting_Compare.change() to compares two monitoring periods against each other.

A. Input Parameters
Lighting Analysis:

- **Load_data**: A timeseries of the load values to be evaluated.
- **Timezone** (Optional): The timezone of the data. If none then the default timezone of the OS is used
- **Sq_ft** (Optional): The square footage in feet of the space being evaluated. If none present then no statistics related to square footage will be calculated.
- **Workdays**: A string representing which of the days of the week are to be considered workdays. Days are encoded as 0-6 where 0 is Sunday. Default is a workweek of Mon-Fri which is represented as "12345"
- **workday_start**: A representing the start of the workday. See the LightingSummary.R notes for a section of implementation and how this affects non-standard workplaces like nightclubs.

Lighting Compare:

- **baseline_load_data**: A timeseries of load data. Any results will be with respect to this baseline data. I.E. a positive value indicates that the comparison value is greater than the baseline and a negative value indicates that the comparison period is less than the baseline.
- **comparison_load_data**: A timeseries representing the period to be compared to the baseline
- **Sq_ft** (Optional): The square footage in feet of the space being evaluated.
- **Workdays**: A string representing which of the days of the week are to be considered workdays. Days are encoded as 0-6 where 0 is Sunday. Default is a workweek of Mon-Fri which is represented as "12345"
- **workday_start**: A string representing the start of the workday. See the LightingSummary.R notes for a section of implementation and how this affects non-standard workplaces like nightclubs.

B. Input Formats

- **load_data**: It may be a list of tuples containing timestamps and values, filename of a csv file containing the same or a Series object
- **baseline_load_data**: See load_data.
- **comparison_load_data**: See load_data
- **timezone** (Optional): A string that can be parsed by pytz.timezone (http://pytz.sourceforge.net/).
- **sq_ft**: Convertible to a float or None.
- **workdays**: String of at most 7 characters. Days are encoded as 0-6 where 0 is Sunday. Default is a workweek of Mon-Fri which is represented as "12345"
- **workday_start**: A string of the form hour:minute. In strftime notation "%H:%M"

C. Output

Lighting_Analysis returns a dictionary with an entry called "summary_stats." This "summary_stats" item will have the fields and values that are returned from the lighting_baseline module. Current values are "nonworkdaynightmean", "holidaynightmean",

"nonworkdaydaymean", "nonholidayweekmean", "workdaynightmean", "workdaydaymean", and "holidaydaymean"
Lighting_Compare returns a dictionary with one item called “comparison_stats”. This “comparison_stats” item has the same fields as the return from Lighting_Analysis. As noted above a positive value in the comparison stats means that the comparison period was that much more that stat than the baseline while a negative indicates it was that much less than the baseline.

lighting_fdd Execution Instructions

This program is intended to be used as a standard python module and is used by importing lighting_fdd, creating a Lighting_FDD object and calling get_faults() to perform the analysis.

A, Input Parameters

Required:

- **implemented_schedule**: The schedule the space should be following. Any faults will be with respect with this schedule.
- **relay_timeseries**: A timeseries indicating the state of the lights in the zone.
- **override_timeseries**: A timeseries representing the occurrences of manual overrides.
- **override_timeout** (default 2 hours): The programmed length of the override timeout. The assumption is that once an override has occurred the system will remain in the override state (absent any other factors) for the length of the override_timeout.

  For example, if the override_timeout is 2 hours and the system is scheduled to be off but an override happens the system will be expected to be on for the next 2 hours and then off again. So (again absent anything else) if the system is off during the next two hours or on after the elapse of the timeout that is a fault. But if the system remains on for two hours and then turns off there is no fault, that is the override working as expected.

Optional:

- **load_timeseries** (default None): If there is available load data the program can make use of it to see if the change in load matches the change in relay state further refining any diagnostic messages.
- **expected_load_min_change** (default None): If there is load it is likely the case that the load is not purely for the lighting systems or that the measurements do not go to 0 when the relay is off. This value (and the following expected_load _max_change) value allow for an expected load change range. If the load of the system is roughly known then it can be entered to increase the accuracy of diagnostics made from the load_timeseries.
- **expected_load_max_change** (default None): See expected_load_min_change
- **time_zone** (default None) – The timezone of the data. If none is passed the default system timezone will be used.
- **relay_time_comparison_epsilon** (default 5 minutes) – It is often the case that some of the streams get slightly out of sync or have some minor time drift. This value allows for

defining a window around scheduled points so that the system change does not have to fall exactly on the scheduled time. If the system exits the window around the schedule point in the correct state but there was some point inside the window that is in the incorrect state those points are ignored.

For example assume the lights are scheduled to turn off at 18:00:00, the relay_time_comparison_epsilon is 5 minutes, and the relay timeseries looks like [(17:59:00, 1), (18:00:00, 1), (18:01:00, 1), (18:02:00, 1), (18:03:00, 0), … (18:06:00, 0)]

Then in that case there would be no reported fault. If there were no relay_time_comparison_epsilon or if it were set to 1 minute there would be a fault because the relay did not turn off until 18:02 which is 2 minutes after the scheduled time.

- **intended_schedule** (default None) – It may be the case that the implemented schedule is read from the system while there is some other representation of what the intended schedule is (a design document or the like). In case they differ in a meaningful way the application can provide information about the difference as well as locate faults with respect to the intended schedule as well as the one that is actually implemented. This can be useful for distinguishing between hardware faults and system misconfigurations.
- **system_time_vs_real_time_timeseries** (default None) – Clocks can drift. If there is some more accurate ground truth and how it relates to the points recorded in the schedule the application can identify it and report any issues accordingly.
- **compress_overrides_to_change_of_state** (default True) - In our system the override data stream was polled rather than interrupt-driven so it was necessary to take the data from that stream and only use the change-of-value points. However other systems that may have interrupt-driven overrides. To accommodate this set this value to False. Then each point in the override stream will be treated as a separate override event.

  For example, suppose in our system the override switch state is polled each second and remains in its current state until some other event. So if there was an override to off at 20:00:00 the override stream would look like [(19:59:58, 1) , (19:59:59, 1), (20:00:00, 0), (20:00:01, 0), (20:00:02, 0), …]. In order to make sense of this the program first determines that the actual override event happened at 20:00:00 and uses this in the diagnosis.
  In a second system suppose that the override is an interrupt and only registers that value. In that system if there is an override to on at 20:00:00 and another override to on at 22:00:00 that override stream would look like [(20:00:00, 1), (22:00:00, 1)]. Under the scheme in the paragraph above the second override would be ignored. Setting the compress_overrides_to_change_of_state flag to False would make each point be treated as a separate override and would give the correct analysis in this second system.

B. Input Format

- **Timeseries**: Since there are a variety of ways to express this concept the app attempts to be permissive and allows the following forms:
  1. A path to a file containing the timeseries in csv form where each (timestamp, value) pair is on one line and the timestamp value is convertible using the dateutil.parser.parse function (https://labix.org/python-dateutil#head-c0e81a473b647dfa787dc11e8c69557ec2c3ecd2).
     a. Example: "my_timestamp_file" where the file in question has contents that look like "May 19, 2014 00:00:00, 1\nMay 19, 2014 00:00:05, 1.2\n etc…"
  2. A string containing the same content structure as the csv file described in 1
     a. Example: "May 19, 2014 00:00:00, 1\nMay 19, 2014 00:00:05, 1.2\n etc…"
  3. A list where each entry has the same format as described in 1.
     a. Example: ["May 19, 2014 00:00:00, 1", "May 19, 2014 00:00:05, 1.2", etc…]
  4. A list of pairs where the first item in the pair is either a UNIX time or convertible to a datetime using dateutil.parser.parse.
     a. Example: [(1400482800000, 1), (1400482805000, 1.2), etc…]
  5. A list of pairs where the first item is a datetime.datetime object and the second is a float.
     a. Example [(datetime.datetime(2014, 5, 19, 0, 0, 0), 1), datetime.datetime(2014, 5, 19, 0, 0, 5), 1.2)]
- **implemented_schedule**: see Schedule in "Additional Considerations and Concepts". Must be ether a list of (timestamp, value) pairs or a filename for a csv containing the same.
- **relay_timeseries**: see Timeseries. This is a timeseries representing the measured state of the relay system. In this timeseries values must be either 0 or 1 where 0 is off and 1 is on.
- **override_timeseries**: see Timeseries. This is a timeseries representing the occurrences of manual overrides. While this value is required it is possible that no overrides occurred during the monitoring period. To represent that either pass None as the value or an empty list.
- **override_timeout** (default 2 hours): A timedelta object .

Optional

- **load_timeseries** (default None): See Timeseries. If not None the load values units do not matter so long as they are the same units as expected_load_min_change and expected_load_max_change
- **expected_load_min_change** (default None): A float or int.
- **expected_load_max_change** (default None): A float or int.
- **time_zone** (default None): A pytz timezone object or a string in a form that can be parsed by pytz.timezone (http://pytz.sourceforge.net/). If none is passed the default system timezone will be used.
- **relay_time_comparison_epsilon** (default 5 minutes): A timedelta object.
- **intended_schedule** (default None): see Schedule in "Additional Considerations and Concepts". Must be ether a list of (timestamp, value) pairs or a filename for a csv containing the same.

- **system_time_vs_real_time_timeseries** (default None): A timeseries where the second value is another timestamp. The first timestamp represents the system time and the second represents the "real" time.
- **compress_overrides_to_change_of_state** (default True): A bool

C. Output

Lighting_FDD.get_faults() performs the analysis and returns four values: faults with implemented schedule, faults with intended schedule, suggested schedule changes and suggested override timeout changes. All are lists where each entry is one instance of the specific fault.

Faults: Each fault response (implemented and intended) is a list (or None if there are no faults of that type.) Each fault will have the following explanatory fields:

- Fault description – A message that describes the fault and, if possible, provides some indication of likely locations for the fault.
- Estimated start time – The time the fault appears in the stream. Note that this may not correspond to the actual time the fault starts, it is just the first time that it is seen in the data.
- Estimated stop time – The time when the system re-enters a non-faulty state. Note that depending on the nature of the fault the fault may still exist (e.g. if the error is the system does not turn off when scheduled the end time will be the next time the system is scheduled to be on) or the fault may truly be over (e.g. if there was a glitch in recording the data due to a power outage).

There are (potentially) two lists of faults the program returns. Comparing them can help diagnose whether there is a problem with a component in the system or if the issue is misconfiguration.

Suggested changes: Unlike faults suggested changes do not apply to a specific date and time. They are system in nature and are meant to convey information about how to tune the system as a whole. They come in two main categories: Efficiency and User Experience. Broadly speaking when the app can detect that the lights are being left on unnecessarily it will suggest measures that could reduce the total usage of the lights while maintaining user experience. Likewise when it can detect that the lights are scheduled to be off when the space is occupied it can suggest measures to try to increase occupant satisfaction.

Currently there are two schedule changes the program can suggest:

1. Potential to improve efficiency by reducing the scheduled hours-on.
2. Potential to improve service by extending scheduled hours-on.

And two override changes:

1. Potential to improve efficiency by reducing the length of the override timeout.
2. Potential to improve service by extending the length of the override timeout.

D. Notes

As of this writing there is only the weekly schedule so care needs to be taken when using actual datetime objects to use as a schedule. Since only the day of week is taken any attempt at defining a schedule that spans more than one week and has different behavior on the same weekday will produce unexpected results. In short when creating a weekly schedule the app accepts datetimes as inputs but only uses the datetime.weekday() and datetime.time() methods to create the schedule, not the .date() function. So any entries that share the same weekday() value but differ in date() values will be used to create the same schedule day.
For example a schedule that had the following entries
"2014-09-01 08:00:00, 1
2014-09-01 18:00:00, 0
…
2014-09-08 07:00:00, 1
2014-09-08 17:00:00, 0"

Would generate a schedule for Monday that has the lights scheduled to turn on at 07:00:00 and off at 17:00:00. This is because the schedule's entries for "Monday" look like:
[(7:00:00, 1), (8:00:00, 1), (17:00:00, 0), (18:00:00, 0)] Since the lights are already supposed to be on at 7 turning them on again at 8 has little effect (aside from possibly adding to the reported fault count.). Similarly for turning them off at 17:00:00 then turning them off again at 18:00:00.
Execution Instructions for using volttron agents
See https://github.com/VOLTTRON/volttron/wiki for instructions running Volttron agents.

A. LightingBaselineAgent Execution Instructions
Topics this agent subscribes to:
"lighting_baseline/summary_request"
"lighting_baseline/comparison_request"

These correspond to the summary and comparison requests mentioned above.

Summary_request:
- Header fields:
  - Required:
    - requestorID: The ID of the requesting agent. Needed to craft the response topic
  - Optional:
    - **From**: Will be used to set the "To" field in the response headers if present
- Message fields:
  - Required:
    - **load_data**: A timeseries of the load values to be evaluated
  - Optional:
    - **timezone**: The timezone of the values in load_data. If none is present then the OS timezone will be used.
    - **sq_ft**: The square-footage of the space being evaluated.

        - **workdays**: A string representing which of the days of the week are to be considered workdays. Days are encoded as 0-6 where 0 is Sunday. Default is a workweek of Mon-Fri which is represented as "12345"
        - **workday_start**: A string representing the start of a 12-hour workday. Default is "6:30".
- Response Topic: "lighting_baseline/summary_response/{requestor_ID}" where requestor_ID is the value in the initial request header's "requestorID" field. For example, if the requestorID value was "CustomBuildingManagementAgent" the response topic would be "lighting_baseline/summary_response/CustomBuildingManagementAgent"
- Response Headers:
    - To: The value in the "From" header field from the initial request if it exists, "Unknown" otherwise.
    - **From**: This will always be "lightingAnalysisAgent"
- Response Message: A dictionary containing the results of the lighting_baseline module calculations. Current results are listed as "summary_stats" and they are: "nonworkdaynightmean", "holidaynightmean", "nonworkdaydaymean", "nonholidayweekmean", "workdaynightmean", "workdaydaymean", and "holidaydaymean". Consult the lighting_baseline module documentation for additional information on these results.

Comparison_request:
- Header fields:
    - Required:
        - **requestorID**: The ID of the requesting agent. Needed to craft the response topic
    - Optional:
        - **From**: Will be used to set the "To" field in the response headers if present
- Message fields:
    - Required:
        - **baseline_load_data**: A timeseries of the load values representing the baseline to which the comparison will be made. Any results will be with respect to this baseline data. I.E. a positive value indicates that the comparison value is greater than the baseline and a negative value indicates that the comparison period is less than the baseline.
        - **comparison_load_data**: A timeseries of the load values representing the period to be compared to the baseline.
    - Optional:
        - **timezone**: The timezone of the values in load_data. Must be a string that is parsable by pytz.timezone (http://pytz.sourceforge.net/). If none is present then the OS timezone will be used.
        - **sq_ft**: The square-footage of the space being evaluated.
        - **workdays**: Which of the days of the week are to be considered workdays. Default is a workweek of Mon-Fri which is represented as "12345"

        - **workday_start** (default "6:30): The start of a 12-hour workday.
- Response Topic: "lighting_baseline/comparison_response/{requestor_ID}" where requestor_ID is the value in the initial request's "requestorID" header field. For example, if the requestorID value was "CustomBuildingManagementAgent" the response topic would be "lighting_baseline/summary_response/CustomBuildingManagementAgent"
- Response Headers:
    - **To**: The value in the "From" header field from the initial request if it exists, "Unknown" otherwise.
    - **From**: This will always be "lightingAnalysisAgent"
- Response Message: A dictionary containing the results of the lighting_baseline module calculations. Current results are listed as "comparison_stats" and they are: "nonworkdaynightmean", "holidaynightmean", "nonworkdaydaymean", "nonholidayweekmean", "workdaynightmean", "workdaydaymean", and "holidaydaymean". These numbers are the change from the baseline state. For example, if the workdaydaymean is negative that means that there was less usage in the comparison period during workday days. If it were positive that means that there was less usage in the baseline period during workday days. Consult the lighting_baseline module documentation for additional information on these results.

B. Input Formats

Load data timeseries: A timeseries of load values. load_data, baseline_load_data, and comparison_load data are all examples of fields that are load data timeseries. The timeseries is a list of (timestamp, value) pairs where value is the load value and timestamp may be in any of the following formats:

1. Seconds since 1970-01-01 00:00:00
2. Milliseconds since 1970-01-01 00:00:00
3. Year-month-day hours:minutes:seconds, e.g. 2014-08-28 16:30:00

The program automatically determines what time format is being used: if the first timestamp in the file includes a ":" then the program assumes format 3 is being used. If it does not, then numbers exceeding 3×10^9 are assumed to be in milliseconds, otherwise they are assumed to be in seconds. Timestamps do not need to be at any particular frequency and do not need to fall on round numbers of minutes or hours. They are assumed to be regularly spaced in time; if they are not, no error is generated but the program's results may be incorrect.

timezone: A string that can be parsed by pytz.timezone (http://pytz.sourceforge.net/).

sq_ft: Convertible to a float or None.

workdays: String of at most 7 characters. Days are encoded as 0-6 where 0 is Sunday. Default is a workweek of Mon-Fri which is represented as "12345"

workday_start: A string of the form hour:minute. In strftime notation "%H:%M"

C. Output

The agent responds to any output by posting a message to the output topic that corresponds to the topic the request was made on. For a summary request the response will be posted to

“lighting_baseline/summary_response/{requestor_ID}” while for a comparison request that will be “lighting_baseline/comparison_response/{requestor_ID}”
If there was a problem with the request the agent will respond with a message dictionary that has one entry called “error” where the contents are a description of the error.
If there was no problem with agent will post the results of the analysis. Currently the available results are: "nonworkdaynightmean", "holidaynightmean", "nonworkdaydaymean", "nonholidayweekmean", "workdaynightmean", "workdaydaymean", and "holidaydaymean”.
For information about what exactly these numbers mean please consult the lighting_baseline module documentation.

D. Output Formats
The values in the result fields will either be convertible to a float or “Nan”.

DISCLAIMER

This document was prepared as an account of work sponsored by the United States Government. While this document is believed to contain correct information, neither the United States Government nor any agency thereof, nor the Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or the Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or the Regents of the University of California.