SAE INTERNATIONAL®

Integration of OpenADR with Node-RED for Demand Response Load Control Using Internet of Things Approach

2017-01-1702
Published 03/28/2017

Piyush Aggarwal and Bo Chen
Michigan Technological University

Jason Harper
Argonne National Laboratory

CITATION: Aggarwal, P., Chen, B., and Harper, J., "Integration of OpenADR with Node-RED for Demand Response Load Control Using Internet of Things Approach," SAE Technical Paper 2017-01-1702, 2017, doi:10.4271/2017-01-1702.

Abstract

The increased market share of electric vehicles and renewable energy resources have raised concerns about their impact on the current electrical distribution grid. To achieve sustainable and stable power distribution, a lot of effort has been made to implement smart grids. This paper addresses Demand Response (DR) load control in a smart grid using Internet of Things (IoT) technology. A smart grid is a networked electrical grid which includes a variety of components and sub-systems, including renewable energy resources, controllable loads, smart meters, and automation devices. An IoT approach is a good fit for the control and energy management of smart grids. Although there are various commercial systems available for smart grid control, the systems based on open sources are limited. In this study, we adopt an open source development platform named Node-RED to integrate DR capabilities in a smart grid for DR load control. The DR system employs the OpenADR standard. To enable an OpenADR Virtual Top Node (VTN) to communicate with Node-RED nodes, an OpenADR Node-RED node has been developed to translate and pass information between an OpenADR VTN and Node-RED nodes. The implementation is validated for the use cases involving charging control of plug-in electric vehicles in response to real-time pricing from a utility. It has been observed that the data transfer is taking place in accordance with the standard and the Plug-in Electric Vehicle (PEV) charging is coordinated with real-time pricing and desired target State of Charge (SOC).

Keywords

Node-RED, Node.js, OpenADR, VEN, VTN, Demand Response, Load Control, IoT, Internet of Things

I. Introduction

With the rising concern of sustainable mobility, a lot of effort is being made to introduce Plug-in Electric Vehicles (PEVs) into market. The increasing PEV adoption rate, however, has raised a concern of electrical load congestion on power grids. To overcome this issue, smart grid is considered to be a prominent solution, as it will allow a reliable and transparent control of load balancing. Smart grid allows users to opt when to use electricity and in what amount. At the same time, the utility can maintain load balance by controlling the price of electricity [1][2] or through demand-based load control. To support the wide-scale functionality of smart grid, various standards and protocols are either under development or have been already developed. One such open standard for the communication of demand response signals between a utility and end devices is OpenADR [3]. It standardizes the client server based relationship over Hyper Text Transfer Protocol (HTTP) requests, and makes sure that large-scale interoperable devices can be designed which would motivate the use of this technology [4].

One of the publication by Lawrence Berkeley National Laboratory shows that various pricing models, such as real-time price and time of use, can be successfully implemented as a demand response system [5]. Though the possibility of various models is discussed, the author has not compared any of these methods. Liu et al. [6] have explained in their paper that IoT technology is growing rapidly across all sectors and thus its integration with smart grids is most favorable approach to promote smart energy. For the application of demand response in PEV charging, research [7] has proved that by use of demand response framework, load balancing burden can be pushed towards end users by means of pricing. The author has used a congestion pricing concept to study the system analytically and using simulations. Bui et al. [8] have termed the web enabled smart grid as Internet of Energy and have concluded that it is a key component to ensure practicality of smart grid and help to reach its true potential. Researchers at Alcatel-Lucent have introduced a cloud-based demand response infrastructure on which OpenADR is based [9].

Even though the OpenADR standard is openly available, it requires a lot of programming effort to be implemented in real hardware devices which makes it difficult for Scientists with minimal computer background. To overcome this problem, this paper aims to develop an easy to implement OpenADR client software extension for a Graphical User Interface (GUI) based open source platform named

Node-RED. For the past few years, an enormous amount of growth is observed in evolution of Internet of Things (IoT) based hardware and software platforms [10]. Most suitable platform identified for our case is Node-RED [11]. Node-RED is a widely accepted tool initiated by IBM for the development of IoT devices. Node-RED is a GUI-based tool and thus does not require a thorough knowledge of computer programming for IoT device development. Node-RED being open source and easy to learn, availability of OpenADR extension for such a platform enables the easy and reliable design of customized smart-grid connected devices.

This paper is organized as follows. Section II of this paper introduces basic concepts of OpenADR and Node-RED. Section III presents the implementation details of OpenADR Virtual End Node (VEN) node in Node-RED. Section IV focusses on the development of cost optimized PEV charging control by the use of OpenADR VEN node. In section V, a use case of the charging controller developed in former section is discussed. The limitations and future scope of the development in this field are discussed in section VI. The literature is finally summarized in Section VII.

II. Node-RED-based Demand Response System

Demand Response of a system is defined as the change in electrical consumption by customers as a result of events published by a facility provider. Automated and time-bound implementation of this concept is termed as Automated Demand Response (ADR). For the purpose of this research, a globally accepted Demand Response communication standard named OpenADR is used. Figure 1 explains the model of information transfer for an automated demand response system.

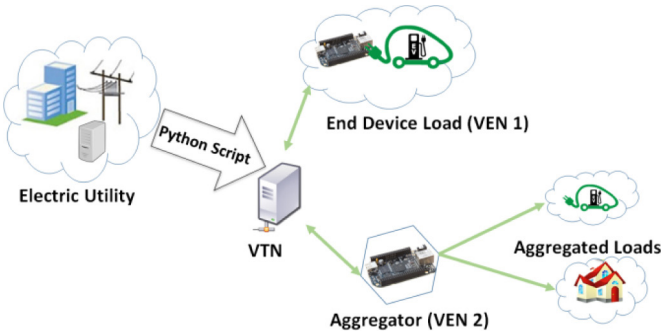

Figure 1. Overview of an Automated Demand Response system

For the system shown in Figure 1, the main components are:

1. Electrical Utility
2. Communication System (OpenADR-based Server and Client)
3. End Device and its controller (eg. EV and EV charger)

A VTN usually located at the electrical utility, sends the DR commands such as the information of electricity price to the client (VEN). The VEN passes the information to an end device controller (in our case EV charge controllers) to decide how much electricity is to be consumed.

As mentioned in Section I, the main focus of this paper is to develop an OpenADR VEN which is based on platform named Node-RED. OpenADR and Node-RED are explained briefly further in this section.

OpenADR

The development of the OpenADR standard started in 2002 at Lawrence Berkeley National Laboratory as a result of the California energy crisis. The design of OpenADR is based on a server/client architecture with xml message transfer over HTTP or XMPP services. It is basically an application layer communication standard for the interactions between the power utility and end devices. The server, generally utility, is termed as VTN and the client is termed as VEN [4]. As per the standard, the events are published by VTN and are communicated to VEN as xml messages. VEN responds to the events by varying the electricity consumption and sends its opt-in/opt-out decision back to the VTN. The specification also allows the transfer of reports which may act as feedback to the utility for load control. Message transfer can take place through two mechanisms, PUSH or PULL. PUSH is similar to broadcasting of event messages from VTN to VEN; however, PULL requires periodic polling VTN to check for the availability of any update.

VTN

As described earlier, a server who publishes all DR events is referred to as the VTN of an OpenADR communication system. For the purpose of this research, an open source version of a VTN [12] developed by EPRI is used.

VEN

In OpenADR communications, the VEN must support either one of the transport protocols among simple HTTP and XMPP. The VEN is responsible for receiving the events from the VTN securely and then responding with appropriate feedback such as an Opt response or report [4]. The VEN may either be a separate device to communicate with VTN and forward the messages to end devices or can be integrated with the end devices.

OpenADR2.0b supports four services: EiRegisterParty service, EiReport service, EiEvent service, and EiOpt service. The EiRegisterParty service is used to register the VEN with the VTN. This is the primary service required for all other communications to happen between the VTN and VEN. Also, if Transport Layer Security (TLS) is implemented, then initial authentication occurs in this step to authorize the VEN to participate in further communications with the VTN. However, for this paper no such security is used and thus this process is used to just initiate the handshake between the VTN and VEN. The EiReport service enables the sharing of data among the VTN and VEN in the form of reports. The EiEvent service defines the message flow between the VTN and VEN for polling and transfer of event payload in the form of xml message. The EiOpt service is used to make the VTN aware of whether the end device is taking part in an event or not. The interactions between the VTN and VEN for these services are summarized in Table 1.

Table 1. The interactions of VTN and VEN for individual services

Services	Message	Direction
EiRegisterParty	oadrQueryRegistration	VTN <----- VEN
	oadrCreatedPartyRegistration	VTN -----> VEN
	oadrCreatePartyRegistration	VTN <----- VEN
	oadrCreatedPartyRegistration	VTN -----> VEN
EiReport	oadrRegisterReport	VTN -----> VEN
	oadrRegisteredReport	VTN <----- VEN
EiEvent	oadrRequestEvent	VTN <----- VEN
	oadrDistributeEvent	VTN -----> VEN
	oadrCreatedEvent	VTN <----- VEN
	OadrResponse	VTN -----> VEN
EiOpt	oadrCreateOpt	VTN <----- VEN
	oadrCreatedOpt	VTN -----> VEN
	oadrCancelOpt	VTN <----- VEN
	oadrCancelledOpt	VTN -----> VEN
Polling	oadrPoll	VTN <----- VEN
	oadrRegisterReport	VTN -----> VEN
	oadrRegisteredReport	VTN <----- VEN
	oadrResponse	VTN -----> VEN
	oadrPoll	VTN <----- VEN
	oadrDistributeEvent	VTN -----> VEN
	oadrCreatedEvent	VTN <----- VEN
	oadrResponse	VTN -----> VEN

Node-RED

Node-RED is a GUI-based IoT software development tool which works on the Node.js JavaScript environment. It is an open source platform and is thus freely available for all operating systems including Linux, Mac OS and Windows. Being an open source project, a strength of this platform is that, nodes (extension libraries) can be developed and published globally using the Node.js package manager (npm) repository. Taking advantage of this opportunity, we have developed a node extension for the OpenADR Pull type VEN which can be easily configured with VTN credentials. The implementation details of this OpenADR VEN node is presented in the next section. Developed Node-RED flow for OpenADR VEN is shown in Figure 2.

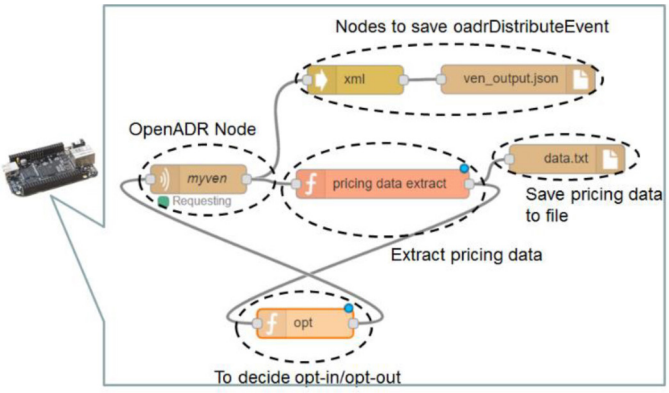

Figure 2. Node-RED flow for OpenADR2.0 VEN

Figure 4 displays the configuration options available for this node. The Name entry specifies the VEN name; URL is the server VTN address; Poll Rate defines the polling rate to pull information from VTN in seconds; the VEN Id is the identification number of the VEN which is pre-registered with VTN. This node allows the communication of both A and B profile of OpenADR 2.0 thus an option is provided to select an appropriate one [13] [14].

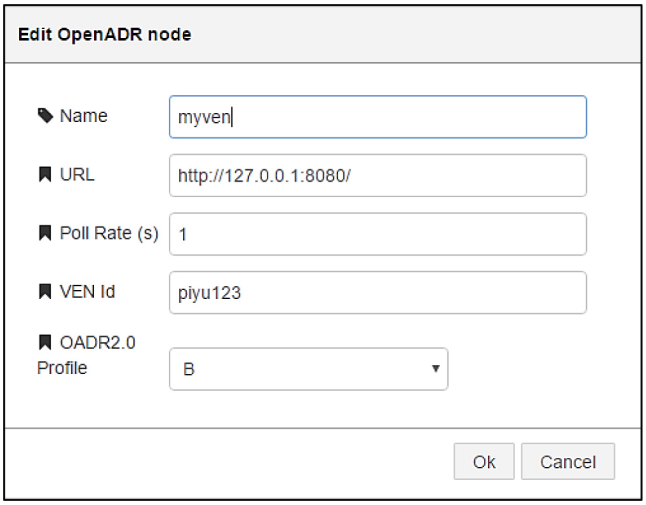

Figure 3. Node-Red OpenADR VEN Node configuration window

III. Node-RED OpenADR VEN Node

The OpenADR specification version followed for this paper is 2.0b. This paper chooses to implement the pull-only type VEN in which case a periodic polling to VTN is required for updated event signals. To fulfill the VEN functions, the components involved in the OpenADR VEN node are shown in Figure 4, which includes functions for the communication with the VTN, message handling and a database to store message details. Currently, only HTTP communication and some message handling functions are implemented in the OpenADR VEN node. The HTTP communication block contains the functions for sending and receiving data between the VTN and VEN through HTTP. The message transfer takes place with an 'application/xml' header through the POST service. The response messages received are handled in the message handling subsystem which extracts the useful information from the xml script and uses it accordingly. In the ideal case, though not implemented in this case, extracted data such as signal values, request identities, registrations id etc. must be stored in a database so that it can be accessed or updated whenever required. Since a database is a crucial part for the appropriate implementation of the complete OpenADR specification, it is planned for future development.

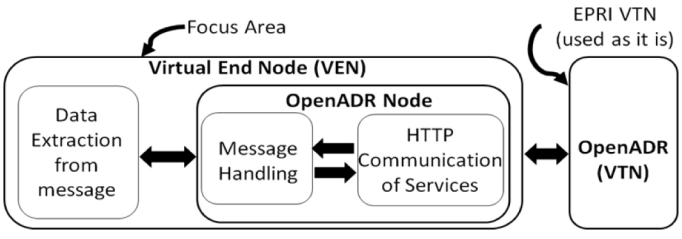

Figure 4. Functions of a VEN node

The implemented communication, services, and message handling for the corresponding services in the OpenADR VEN node are shown in Figure 5.

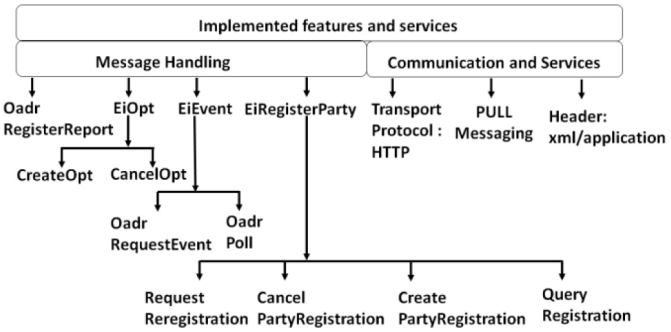

Figure 5. Implemented services and message handling in OpenADR VEN node

As shown in Figure 2 and Figure 4, the main components of Node-RED VEN are:

a). OpenADR-VEN Node
b). Data Extraction Node
c). Opt Node

Development of all highlighted sub components is discussed below.

Development of OpenADR-VEN Node

This node communicates directly with VTN over HTTP protocol as per OpenADR2.0 specification. This is the main node which handles all services as in Figure 5 and receives all the related event information from VTN. For the development of this node, software code is to be written into two files named: 'OpenADR.html' and 'OpenADR.js'. As can be seen from Figure 6, the HTML file contains the code for aesthetics part of this node and its configuration options related to GUI interface in Node-RED window however, the '.js' file contains the code for main functionality such as communications and xml message handling.

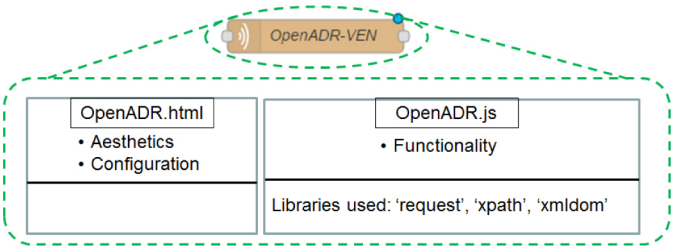

Figure 6. Code Components of OpenADR-VEN Node

The OpenADR VEN node is written in Node.js and uses the following open available libraries named 'request', 'xpath' and 'xmldom'. The 'request' library is used for http communication with the VTN while the other two libraries are used to process the xml messages sent to or received from the VTN. In addition to communicating with a VTN, a VEN node also needs to interact with end devices to pass information between VTN and end devices. For oadrDistributeEvent message received from the VTN, the OpenADR VEN node outputs the message payload values to the end devices.

This OpenADR output message is passed to the end devices in the JSON 'msg.payload' object. Figure 7 represents the program flow chart for the functions written in the file named 'OpenADR.js'.

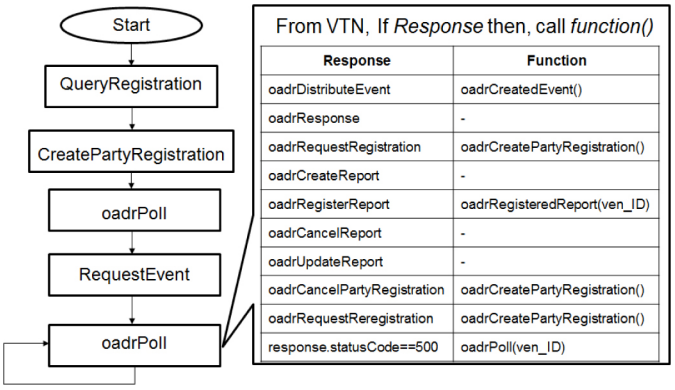

Figure 7. Program flow and functions implemented in OpenADR-VEN Node

Development of Data Extraction Node

The OpenADR VEN node outputs the payload values for the oadrDistributeEvent messages as this message contain the main signal data required for control of end devices. Other message's payload values are visible in Node-RED debug console window. Figure 8 represents the flow chart in accordance to which the code for data extraction node is written. It basically parses the xml message of 'oadrDisrtibuteEvent' payload and extracts the data such as signal name, signal type, signal value, event expiration time etc. The extracted data are then sent to EV charging controller to decide the current charging rate.

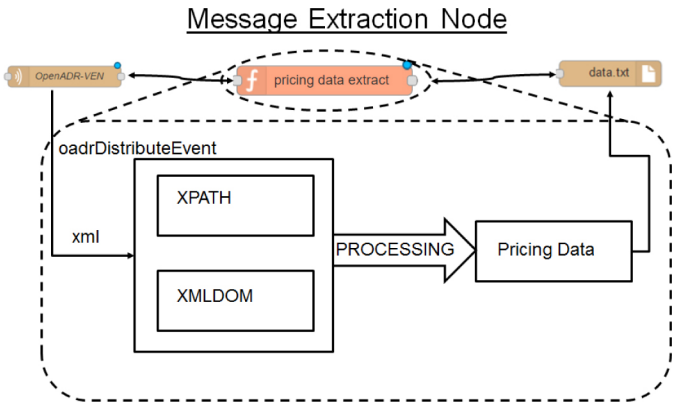

Figure 8. Flow chart for code of data extraction node.

Development of Opt Node

The OpenADR VEN node also contains an input port. This input port receives the Opt-in / Opt-out decision by the end devices or control logic. This opt input must be provided as xml string as shown in Figure 9. In this xml string, data written on bold characters depend on the decision of an end device and is thus replaced with corresponding relevant values.

```
<?xml version="1.0" encoding="utf-8"?>
<oadrPayload xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://openadr.org/oadr-2.0b/2012/07">
<oadrSignedObject>
<oadrCreateOpt d3p1:schemaVersion="2.0b" xmlns:d3p1="http://docs.oasis-
open.org/ns/energyinterop/201110">
<d3p1:optID>OPTID</d3p1:optID>
<d3p1:optType>opttype</d3p1:optType>
<d3p1:optReason>REASON</d3p1:optReason>
<marketContext xmlns="http://docs.oasis-open.org/ns/emix/2011/06" />
<d3p1:venID>VENID</d3p1:venID>
<vavailability xmlns="urn:ietf:params:xml:ns:icalendar-2.0">
<components>
<available>
<properties>
<dtstart>
<date-time>OPT START DATE-TIME</date-time>
</dtstart>
<duration>
<duration>DURATION</duration>
</duration>
</properties>
</available>
</components>
</vavailability>
<d3p1:createdDateTime>'+new
Date().toISOString()+'</d3p1:createdDateTime>
<requestID xmlns="http://docs.oasis-
open.org/ns/energyinterop/201110/payloads">8add0bd9de</requestID>
</oadrCreateOpt>
</oadrSignedObject>
</oadrPayload>
```

Figure 9. XML message for oadrCreateOpt request

The decision of opt type (i.e. Opt-in/Opt-out) is calculated as per the decision chart shown in Figure 10.

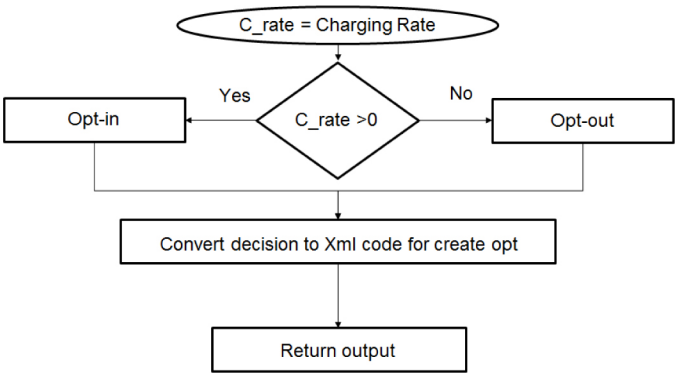

Figure 10. Decision chart for Opt-type (Opt-in/Opt-out)

IV. Implementation of a DR System

To demonstrate the appropriate functioning of the developed OpenADR VEN node, a demand response system was designed. As shown in Figure 11, electricity price is released on ComEd's website [15], which is further published to VTN using a Python script every 5 minutes. If there is any update in the VTN database, it publishes the new price to the related VENs in the form of oadrDistributeEvent. Once the real-time price and forecast price are received, the electric vehicle charge controller calculates the optimal charging rate and passes this value to Electric Vehicle Supply Equipment (EVSE). The control algorithm used to calculate the charging rate is explained further later in this section. For the purposes of illustration, the charging rate is transferred to a Matlab-based model to simulate an electric vehicle battery. In a real application, a J1772 PWM signal can be used to communicate the charge rate to the PEV.

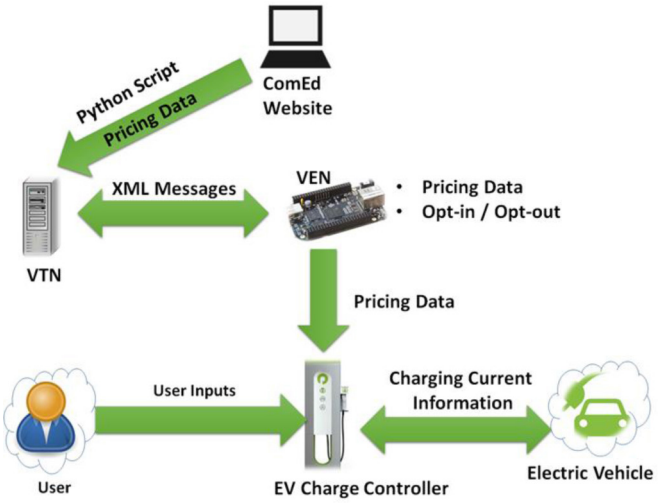

Figure 11. Information exchange among various components

Inputs to EV Charge Controller

As shown in Figure 11, the controller needs inputs from three sources: Utility, User and Electric vehicle. The input data from each source is explained below.

1. Inputs from Users
   a. Desired target SOC: This input is required from the user in terms of percentage. The charging stops when the PEV's SOC reaches the target SOC.
   b. Unplug Time: It is the tentative time at which vehicle will need to be charged to the Target SOC. This allows the controller to calculate charging current.
2. Data from Utility (from ComEd website via VTN)
   a. Electricity Price Forecast: This price schedule is provided by the electric utility. This paper has taken the data from the ComEd web server, which provides the hourly price forecast for the current day and following day. ComEd has reported an improvement in demand response by alerting the customers about price on day-ahead basis [14].
   b. Real-time running hour 5 min average price: ComEd calculates the actual hourly price as average of all 5 min prices in an hour i.e. average of 12 values at the end of an hour. Thus the real-time running hour 5-min average is the average of current hour's 5-min prices.
3. Data from PEVs
   a. Maximum Amperage allowed: This is defined as the maximum charging current that is allowed by the PEV manufacturer. The SAE J1772 standard allows a maximum of 80 Amps for Level-2 charging, however, not all PEVs have the capability to be charged at this value.
   b. Initial SOC of the battery: This is required by the controller to calculate when to stop charging.

Outputs from EV Charge Controller

The controller outputs two main signals, one is the charging rate to EVSE/PEV and the other is the Opt information to the VTN via the VEN. Both of these signals are explained below:

1. Charging Rate:

   The PEV charging rate is sent to EVSE/PEV in the form of 1Khz PWM signal. This PWM is generated as a pilot signal in accordance with the SAE J1772 standard. For the purpose of illustration, the charging rate id transferred to a matlab based model to simulate electric vehicle battery.
2. Opt Information:

   As mentioned earlier in this paper that VTN expects the Opt-in or Opt-out decision from the VEN for the generated events. Thus, the controller sends this Opt decision to the OpenADR VEN node (for transfer to the VTN) in the form of an xml string.

Control Algorithm

The logic used for charging cost optimization is to charge the vehicle during the hours when the electricity price is low and also meet the target SOC requirement. It has been considered that electric utility provides the real-time electricity price and price forecast for current day and following day. For this application, the actual data for both kind of prices are taken from electricity supplier of Chicago, ComEd's website [15] [16].

Mathematically, each hour of the current day and following day (until the vehicle unplug time) is ranked on basis of electricity price, i.e. hour with least price is kept at top (Rank=1) and corresponding charging current is calculated. The function of price w.r.t charging current (in Amps) is shown in equation (1). This relationship (Current vs Price) is generated every 5 minutes on the basis of updated $Price_{max}$ + $Price_{min}$ values.

$$i_{calc} = \frac{i_{max}}{Price_{max}} \times [-Price_{realtime} + (Price_{max} + Price_{min})] \quad (1)$$

where,

i_{calc} is calculated charging current values.

$Price_{max}$ is maximum of all prices for charging duration

$Price_{min}$ is minimum of all prices for charging duration

i_{max} is the maximum current in Amps, at which PEV can be charged.

The charging current as calculated above is then used to calculate the time required to reach desired SOC (%) if vehicle is charged constantly at this rate.

$$\Delta SOC(\%) = SOC_{desired}(\%) - SOC_{now}(\%) \quad (2)$$

$$t_{calc} = \frac{(\Delta SOC \times Q_{max})}{(V \times i_{calc} \times 100) \times \eta \times Cos(\phi)} \quad (3)$$

where,

t_{calc} is time (hours) required to achieve desired SOC, if PEV is charged constantly with i_{calc} current.

Q_{max} is the capacity of the PEV battery in Wh.

$SOC_{desired}$ is the final SOC of the PEV battery specified by the user.

SOC_{now} is the current SOC of the PEV battery.

V is the RMS value of source voltage, 240 V for AC Level-2 chargers.

η is Power converter efficiency

ϕ is Power factor

If the t_{calc} is longer than the time left for unplugging of vehicle, then the vehicle is charged at maximum possible charging rate i_{max} for that vehicle. The real-time calculated charging current value is finally sent to the EVSE and communicated to the PEV as a J1772 PWM signal. The flow chart for algorithm can be seen in Figure 12.

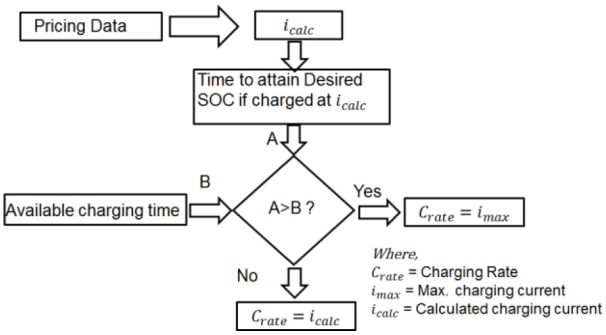

Figure 12. Control Algorithm for charging rate calculation

V. Use Case and Observations

The OpenADR node created for Node-RED is implemented and tested for functionality. Figure 2 shows the created flow for using the OpenADR node and calculating the charging rate using function nodes. It illustrates six nodes of which the main OpenADR client is the one with label 'myven'. It communicates directly with VTN over the internet using http post services. Node with label 'pricing data extract' is the function block and code written within it is used to process xml message received from VTN through 'myven'. Node named 'opt' contain code to send opt information back to VTN through 'myven'. Other nodes are to save the messages in a text file and xml file. A VEN is the combination of all these nodes with 'myven' being the main component through which xml message transaction between VEN and VTN takes place.

To validate the developed OpenADR VEN node, a test case was defined by using the real-time pricing data from electric utility server. The real-time prices provided by the utility are the function of capacity and overall demand from users. The test case is for October11th, 2016 and vehicle is plugged in for charging at 8AM. Table 2 shows all assumptions made for charging rate calculation. Also, simulations has assumed that the AC to DC conversion efficiency is 100% and AC voltage and current are in same phase for all time in equation (3).

Table 2. Assumptions used for the use case

Vehicle	Nissan Leaf SV
Q_{max}	30 KWh
i_{max}	27 A (assumed for 6.6 Kw charging)
Charger Type	Level 2 (240V)

Equation (4) is used to simulate the battery of Nissan Leaf SV as shown in Table 2. Note that $\Delta Charge$ below is different from ΔSOC in equation (2). $\Delta Charge$ equation is used in battery model for calculating change in SOC after each minute of charging.

$$\Delta Charge = 100 - \left[\frac{(Q_{max}/V)-(i/60)}{(Q_{max}/V)} \times 100\right] \quad (4)$$

where,

$\Delta Charge$ is the SOC change of vehicle battery after 1 minute of charging

Q_{max} is the maximum charge capacity of battery in Wh

$\left(\frac{Q_{max}}{V}\right)$ is maximum charge capacity of battery in Ah.

V is the RMS value of source voltage, 240 V for Level-2 charger

i is the current in A, at which battery is being charged.

The final results of the use case are shown in Table 3 and Table 4. In Figure 13, the solid bars represent the real time price variation published by the electric utility and the hashed bars represent the forecasted price for the corresponding hours. In this case the actual charging current and calculated charging current (i_{calc}) are in accordance with the designed algorithm. From Figure 13, it is clear that i_{calc} is varying inversely with electricity price and the actual charging rate is with the consideration of desired SOC. The result shows that the algorithm keeps desired SOC at higher priority as compared to electricity prices.

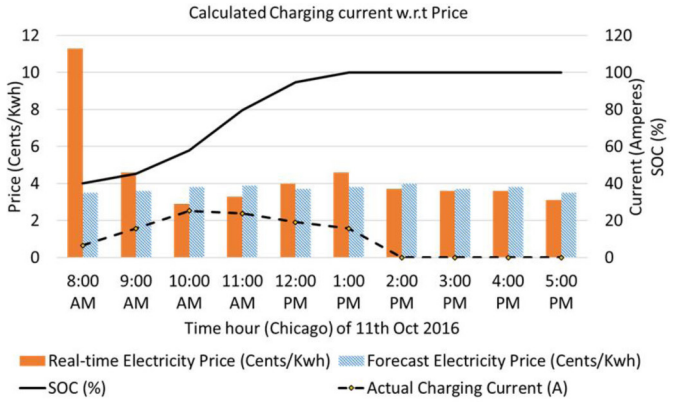

Figure 13. Charging current vs. electricity price calculated for Oct. 11th 2016.

Table 3. Results achieved for the use case

Actual Duration of charging	~ 6 hours
Final SOC achieved	100%

The charging simulation performed for other days and conditions is shown in Table 4. In this figure, the charging cost with real-time prices is compared to the cost if the electricity price is constant at all times. It is clear from the figure that the implemented algorithm ensures the desired SOC within specified time as well as the reduced costs if compared to constant pricing.

Table 4. Vehicle charging results at different times

Test Case	Date (CDT)	Plug / Unplug Time	Actual Charge Duration	SOC (%) (Initial – Final)	*Cost (USD) Real-time-pricing	**Cost (USD) Price-5.11 C/Kwh
1	11-Oct-2016	8:00AM / 5:00PM	~6 hrs	40-100	0.7294	0.9198
2	04-Nov-2016	4:06 PM / 11:00 PM	~3.5 hrs	40-100	0.6178	0.9198
3	05-Nov-2016 / 06-Nov-2016	6:00 PM / 8:00 AM	~5.5 hrs	10-100	0.7675	1.3797
4	05-Nov-2016	6:00 PM / 11:00 PM	~3.2 hrs	40-100	0.5020	0.9198

*Cost is calculated using ComEd's method i.e. using hour ending avg. price for whole hour cost calculation and Capacity Charge = 0.433 cents /KWh [17]
** Electricity Supply Rates only (Does not include transmission charges) [17]

Also, the communication response times between VEN and VTN are observed for Node-RED VEN and compared to the available EPRI VEN. From the comparison shown in Table 5, it is clear that the Node-RED based VTN has better response as compared to EPRI VEN. The reason is that the Node-RED is based on Node.js which allows the non-blocking type of network programming. Table 6 shows the recorded OpenADR communication for developed VEN.

Table 5. Network response time between VEN and VTN

Test Run	Average Network Response Time (s)	
	Node-RED VEN	EPRI VEN
1	0.10	0.26
2	0.006	0.88
3	0.006	0.111

Table 6. Recorded transaction of Node-RED VEN with EPRI VTN

Date / Time (EDT)	Request Type	Response Type	Response Time(s)
2016-11-06 01:30:06	oadrQueryRegistration	oadrCreatedPartyRegistration	0.001
2016-11-06 01:30:06	oadrCreatePartyRegistration	oadrCreatedPartyRegistration	0.003
2016-11-06 01:30:06	oadrPoll	oadrRegisterReport	0.075
2016-11-06 01:30:07	oadrRegisteredReport	oadrResponse	0.005
2016-11-06 01:30:07	oadrRequestEvent	oadrDistributeEvent	0.755
2016-11-06 01:30:08	oadrCreatedEvent	oadrResponse	0.004
2016-11-06 01:30:08	oadrCreateOpt	oadrCreatedOpt	0.002
2016-11-06 01:30:08	oadrPoll	oadrResponse	0.003

The correct operation of the OpenADR VEN node is also validated by observing the message transactions between VTN and VEN. Figure 14 is a screenshot of sample oadrPoll message exchanges between the VEN and VTN, which are in accordance with OpenADR standard. The full list of message exchanges for this test case is shown in Table 6.

```
VEN ----oadrPoll-----> VTN

<?xml           version="1.0"           encoding="utf-8"?>           <oadrPayload
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"        xmlns="http://openadr.org/oadr-
2.0b/2012/07">      <oadrSignedObject>          <oadrPoll  d3p1:schemaVersion="2.0b"
xmlns:d3p1="http://docs.oasis-open.org/ns/energyinterop/201110">
<d3p1:venID>piyu123</d3p1:venID>    </oadrPoll>  </oadrSignedObject> </oadrPayload>

 VTN ----oadrResponse-----> VEN

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns4:oadrPayload                        xmlns="http://www.w3.org/2000/09/xmldsig#"
xmlns:ns2="http://docs.oasis-open.org/ns/emix/2011/06/power"
xmlns:ns3="http://docs.oasis-open.org/ns/emix/2011/06/siscale"
xmlns:ns4="http://openadr.org/oadr-2.0b/2012/07"
xmlns:ns5="http://www.w3.org/2005/Atom"
xmlns:ns6="urn:ietf:params:xml:ns:icalendar-2.0"
xmlns:ns7="http://www.opengis.net/gml/3.2"            xmlns:ns8="http://docs.oasis-
open.org/ns/emix/2011/06"                             xmlns:ns9="http://docs.oasis-
open.org/ns/energyinterop/201110"    xmlns:ns10="http://www.w3.org/2009/xmldsig11#"
xmlns:ns11="urn:ietf:params:xml:ns:icalendar-2.0:stream"
xmlns:ns12="http://docs.oasis-open.org/ns/energyinterop/201110/payloads"
xmlns:ns13="http://openadr.org/oadr-2.0b/2012/07/xmldsig-properties"
xmlns:ns14="urn:un:unece:uncefact:codelist:standard:5:ISO42173A:2010-04-07">
    <ns4:oadrSignedObject>
        <ns4:oadrResponse ns9:schemaVersion="2.0b">
            <ns9:eiResponse>
                <ns9:responseCode>200</ns9:responseCode>
                <ns9:responseDescription>OK</ns9:responseDescription>
                <ns12:requestID></ns12:requestID>
            </ns9:eiResponse>
            <ns9:venID>piyu123</ns9:venID>
        </ns4:oadrResponse>
    </ns4:oadrSignedObject>
</ns4:oadrPayload>
```

Figure 14. Sample of Node-RED console window to view message transaction between VTN and VEN for this test case

VI. Limitations and Future Scope

This paper presents an open source platform for the DR load control of a smart grid connected devices. Demonstration of controlled charging of a PEV was shown. Although the system works acceptably fine, there is still room for the improvement. One such opportunity for improvement is implementation of security. These topics are discussed in detail below.

1. Security: The OpenADR specification mandates the use of Public Key Infrastructure (PKI) based TLS such as SSL and also allows application layer security in terms of xml signatures. Though Node-RED directly allows to enable HTTPS security feature which makes it secure against unauthorized editing of flow but for interaction between VEN and VTN, separate function needs to be added into OpenADR node. When, implementing on low cost, resource constrained embedded devices, use of Asymmetric key may be not advisable as it requires a lot of processing power which may not always be viable for these devices [18].
2. Node-RED is still an evolving Internet of Things' platform and thus a new update rolls out every few days. Some of the very new updates require an updated operating system which may not always be feasible. Thus, a regular monitoring of written code is required till this platform is fully developed.
3. More realistic algorithm for charging control: Scope of this paper is to develop an open source VEN and demonstrate its operation. The algorithms used for EV charging may not be the most efficient ones, however they can be used as a base to develop more robust control logic.
4. Demand Response demonstrations for aggregated loads such as simultaneous charging of multiple EVs can be demonstrated to evaluate the functioning of developed VEN.

A sufficient amount of improvement can be made to the work presented in this paper to provide a robust and secure solution for OpenADR implementation.

VII. Conclusions

This paper is mainly focused to motivate the internet of things based implementation of OpenADR VEN. Node-RED is a GUI-based open source development environment with various communication, sensor, and actuator nodes available. Research on Node-RED based approach for OpenADR is expected to be further continued with fully implemented protocol so that other engineers can integrate OpenADR VEN with load/sensor nodes to develop various demand response load control systems. PEV charging control algorithm developed in this paper is not the most optimal approach, however, it works and successfully demonstrates the automated demand response for EV charging and cost reduction. Major observations to be summarized from this paper are:

1. Developed VEN is independent of operating system as well as hardware platform.
2. Developers with minimal programming experience can develop their own smart grid demand response system with this Node-RED OpenADR VEN..
3. Implemented charging rate control algorithm has shown around 20% reduction in charging cost for real time pricing as compared to fixed pricing of electricity.
4. Communication response time for developed Node-RED VEN is better than that of available EPRI VEN.

References

1. Siano, Pierluig. "Demand response and smart grids—A survey." *Renewable and Sustainable Energy Reviews* 30 (2014): 461–478.
2. Gungor, Vehbi C., "Smart grid technologies: communication technologies and standards." IEEE transactions on Industrial informatics 7.4 (2011): 529–539.
3. McParland, Charle. "OpenADR open source toolkit: Developing open source software for the smart grid." 2011 IEEE Power and Energy Society General Meeting. IEEE, 2011.
4. Alliance, OpenAD. "OpenADR 2.0 b profile specification." Raportti. OpenADR Alliance (2013).
5. Ghatikar, Giris. "Open automated demand response technologies for dynamic pricing and smart grid." Lawrence Berkeley National Laboratory (2010).
6. Liu, Jianming, . "Applications of Internet of Things on smart grid in China." Advanced Communication Technology (ICACT), 2011 13th International Conference on. IEEE, 2011.
7. Fan, Zhon. "A distributed demand response algorithm and its application to PHEV charging in smart grids." *IEEE Transactions on Smart Grid* 3.3 (2012): 1280–1290.
8. Bui, Nicola, . "The internet of energy: a web-enabled smart grid system." IEEE Network 26.4 (2012): 39–45.
9. Kim, Hongseok, "Cloud-based demand response for smart grid: Architecture and distributed algorithms." Smart Grid Communications (SmartGridComm), 2011 IEEE International Conference on. IEEE, 2011.
10. Da Xu, Li, He Wu, and Li Shancan. "Internet of things in industries: A survey." IEEE Transactions on Industrial Informatics 10, no. 4 (2014): 2233–2243.Harvard
11. Node-RED Documentation http://nodered.org/docs/. Accessed 11 Oct 2016
12. EPRI OpenADR-Virtual-Top-Node Manual. In: GitHub. https://github.com/epri-dev/openadr-virtual-top-node/releases/tag/v2.0. Accessed 17 Sep 2016
13. Prehofer, Christian, and Chiarabini Luc. "From Internet of Things Mashups to Model-Based Development." Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual. Vol. 3. IEEE, 2015.
14. Salihbegovic, A., . "Design of a domain specific language and IDE for Internet of things applications." Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2015 38th International Convention on. IEEE, 2015.
15. ComEd's Hourly Pricing Program. https://hourlypricing.comed.com/live-prices/. Accessed 17 Sep 2016
16. Faruqui, Ahmad, Hledik Ryan, and Tsoukalis Joh. "The power of dynamic pricing." *The Electricity Journal* 22.3 (2009): 42–56.
17. ComEd Electricity Prices. https://www.comed.com/MyAccount/MyBillUsage/Pages/CurrentRatesTariffs.aspx
18. Bekara, Chaki. "Security issues and challenges for the iot-based smart grid." Procedia Computer Science 34 (2014): 532–537.

Contact Information

Piyush Aggarwal:
piyushaggarwal1990@gmail.com

Dr. Bo Chen:
bochen@mtu.edu

Definitions/Abbreviations

ADR - Automated Demand Response

DR - Demand Response

EPRI - Electric Power Research Institute

EVs - Electric Vehicles

EVSE - Electric Vehicle Supply Equipment

GUI - Graphical User Interface

HTTP - Hyper Text Transfer Protocol

IoT - Internet of Things

NPM - Node Package Manager

PEVs - Plug-in Electric Vehicles

PWM - Pulse Width Modulation

SOC - State of Charge

TLS - Transport Layer Security

VEN - Virtual End Node

VTN - Virtual Top Node

XML - Extensible Markup Language

XMPP - Extensible Messaging & Presence Protocol

The Engineering Meetings Board has approved this paper for publication. It has successfully completed SAE's peer review process under the supervision of the session organizer. The process requires a minimum of three (3) reviews by industry experts.

ISSN 0148-7191

http://papers.sae.org/2017-01-1702