Fall | 2015

Technical Report

Powernet Team

Enclosed in this document is the technical report of the SLAC (Powernet) sponsored by Stanford (SLAC).

Team Members: Bixian Bao, Praveen Gandala, Yizhe Chen, Cory Pruce

Table of Contents

1. Introduction

The project "Powernet" is an initiative of Stanford Linear Accelerator Laboratory (SLAC) operated by Stanford on behalf of DOE (Department of Energy, US) [1].

The main objective of "Powernet" project is to provide an end-to-end open source technology for economically efficient, scalable and secured coordination of grid resources [1].
The project is being developed in collaboration with Carnegie Mellon University (SV) students, under the guidance of Prof Osman Yagan (Assistant Prof (E&C) at CMU) and Sila Kilicotte, Staff scientist at SLAC [1].

The project was started as a proof of concept to identify the potential use of information technology in the grid system to harness the information/data available and to exploit data driven approach in making intelligent decision.

2. Motivation

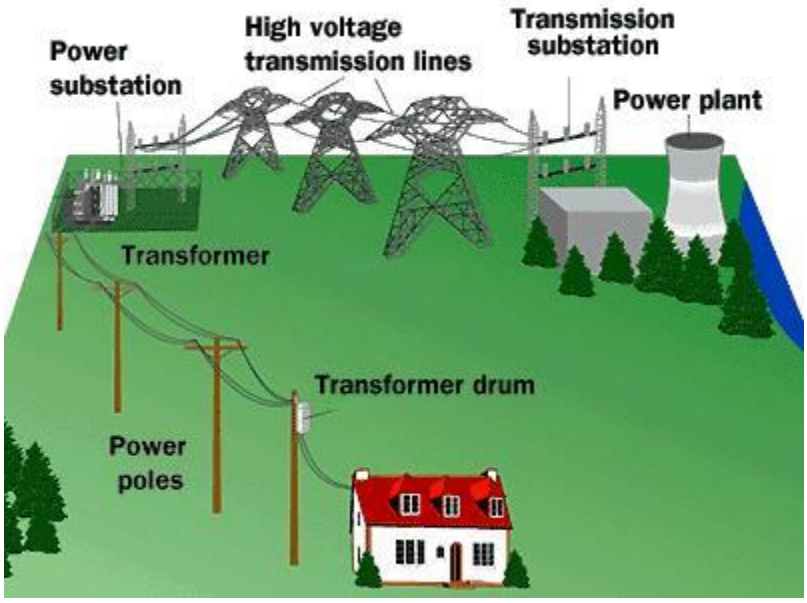
The project is a stepping stone to integrate Information Technology (IT) into the traditional grid system to make it a "smart grid". The need for "smart grid" arises due to the unprecedented change in the ecosystem of electric grid both on the demand as well as on the supply side. Some of these changes are listed below.

1. Proliferation in use of Solar PV & Wind as source of power resulting in two way flow in the electric grid [1].
2. Reduction in cost of storage resulting in their use on a large scale for storing power [1].
3. Emergence of decarburization policies [1].
4. Increase in natural calamities disruption grid services [1].
5. Variation in load due to introduction of electric cars.

2.1. Background

2.1.1. Traditional Grid System

In a traditional grid system power is generated using nonrenewable source of energy such as hydro or thermal and is transferred to the "Power substations" over the high voltage transmission lines. These substations in turn distribute it to the end user. The description provided is a minimalistic overview of a traditional grid system and may be highly complicated (with multiple substation) in the actual scenario.



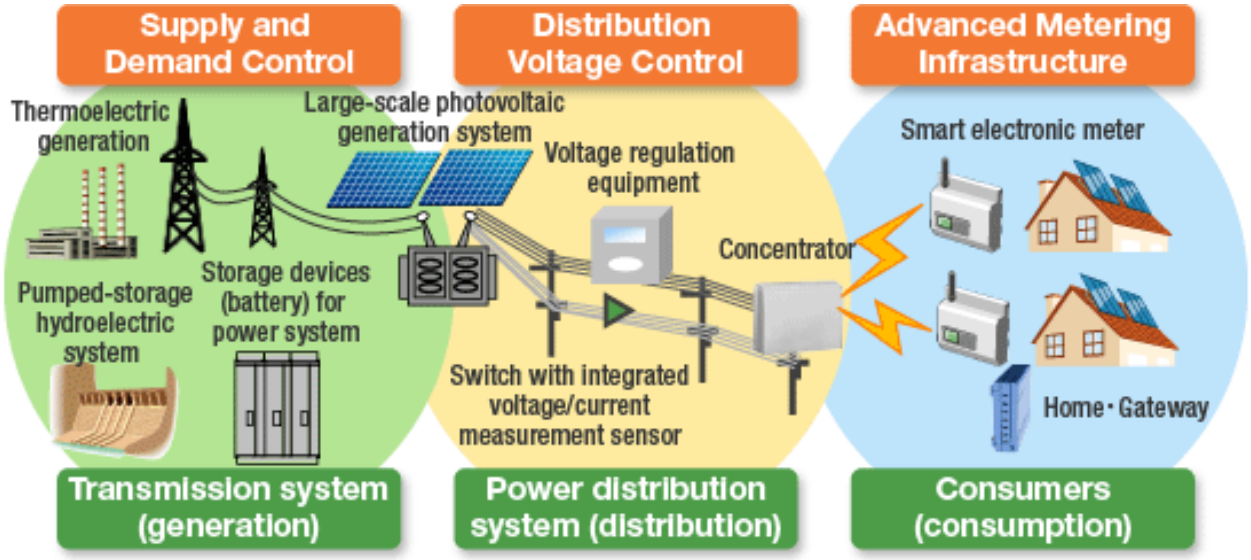
Traditional Electric Grid [2]

Traditional grid system has the following limitations

1. The grid is designed for peak demand. However, the demand tends to vary during the day and the night. Additionally, the variation may be seasonal. This results in wastage of resources.
2. Integration of distributed energy sources such as solar PV, wind energy or storage battery is difficult.
3. It does not support dynamic pricing which can be exploited to implement “Demand Response”, a way to reduce the consumption by incentivizing the end customers, thereby balancing the grid in case the demand exceeds the designed capacity.
4. It does not support information exchange within the grid, which could be used for collecting data to perform advance analytics for e.g. predicting the supply and demand or shortage of electricity in the grid.

2.1.2. Smart Grid System

With a view to address the challenges introduced previously, “smart grid” emerged as the alternate solution. Immense innovation and standardization have emerged and continue to emerge in supply, distribution and metering infrastructure of the grid system.



Smart Grid [3]

Some of the key motivations for a "smart grid" are

1. Self-healing from power disturbance events[4].
2. Enable active participation by consumers in demand response[4].
3. Operational resiliently against physical and cyber attack[4].
4. Improve power quality for 21st century needs[4].
5. Accommodate all generation and storage options[4].
6. Enable new products, services, and markets[4].
7. Optimize assets and operating efficiently[4].

3. Related work

Our focus of study was two aspects of the "smart Grid" ecosystem namely "Demand Response" and "Distributed Energy Resource Management"

3.1. OpenADR Alliance [5]

"OpenADR Alliance is a consortium of members- utilities, device manufacturer, national labs, DR aggregators etc.- actively involved in bringing interoperability in DR (Demand Response) space by proposing OpenADR smart grid standard. Its vision is to standardize, automate and simplify the use of Demand Response (DR) worldwide – making DR a more reliable and cost-effective resource to help utilities meet growing demand for energy, and giving customers greater control over their energy future" [5].

The term "Demand Response" refers to the "changes in electric usage by demand-side resources from their normal consumption patterns in response to changes in the price of electricity over time, or to incentive payments designed to induce lower electricity use at times of high wholesale market prices or when system reliability is jeopardized." [5]

3.1.1. Advantages

The standardization in DR (Demand Response) has proved to be of great value to multiple stakeholders in the electric grid space.

Utilities & Energy Service Providers [5]		
1	Increased grid reliability	By providing more predictable DR resource for utilities
2	Differ need for new generation	DR enables the demand side to vary their consumption thereby differ the need for new power generators
Customers [5]		
1	More control over utility bills	Customers installing DR systems get the dual benefit of managing their consumption based on dynamic cost and also from incentives provided by suppliers
2	Choice of products & vendors	The standardization provides immunity to the customer from vendor locking in use of devices
Equipment Manufacturers & System Integrators[5]		
1	Standardization	Ease of installation and operation
		Reduced complexity

3.1.2. OpenADR Specification

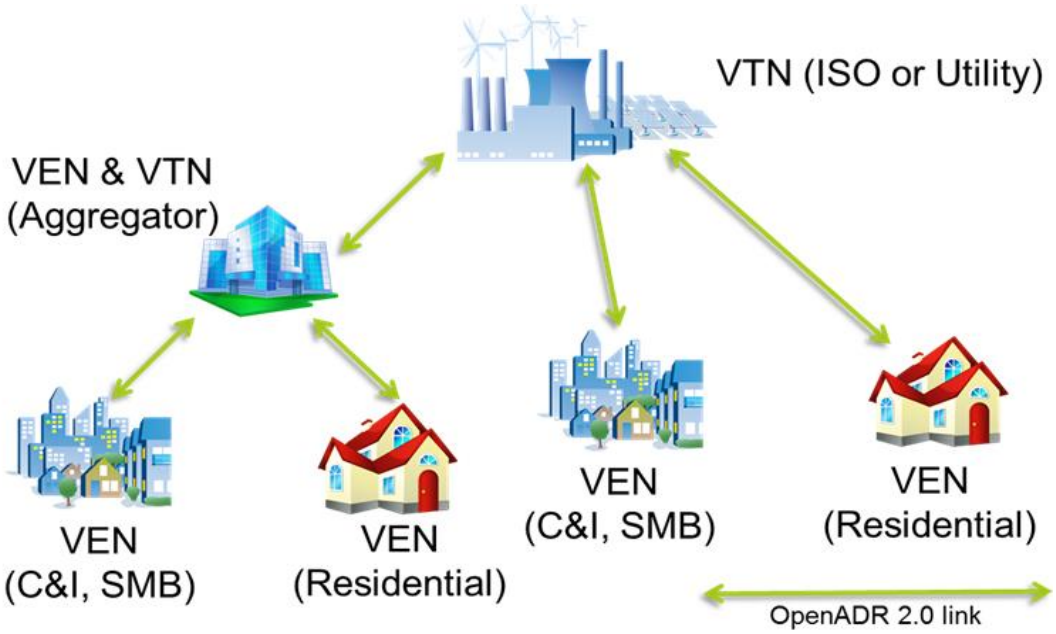
The original OpenADR specification was initially proposed by Lawrence Berkeley National Laboratory (Berkeley Lab). Currently other organizations listed below have joined hands to enhance the specification [5]

1. Organization for the Advancement of Structured Information Standards (OASIS) [5]
2. Utilities Communications Architecture International User's Group (UCAIug) [5]
3. North American Energy Standards Board (NAESB) [5].

3.1.3. OpenADR Architecture

The OpenADR architecture defines the existence of two nodes namely Virtual Top Node (VTN) and Virtual End Node (VEN)

- **Virtual Top Node (VTN)** [5]
  Virtual Top Node is a server responsible for transmitting price information to the other server or terminal/client devices via OpenDER protocol.[5]
- **Virtual End Node (VEN)** [5]
  Virtual End Node is the terminal devices such as thermostat that receives price signals in OpenDER format from The Virtual End Nodes (VEN). "Energy management Systems" can also acts as a VEN. In certain cases same device can act as a VEN and a VTN for e.g. DR aggregation servers can acts as VEN for utility DR(Demand Response) signals or as a VTN for end devices.[5]



VTN and VEN [5]

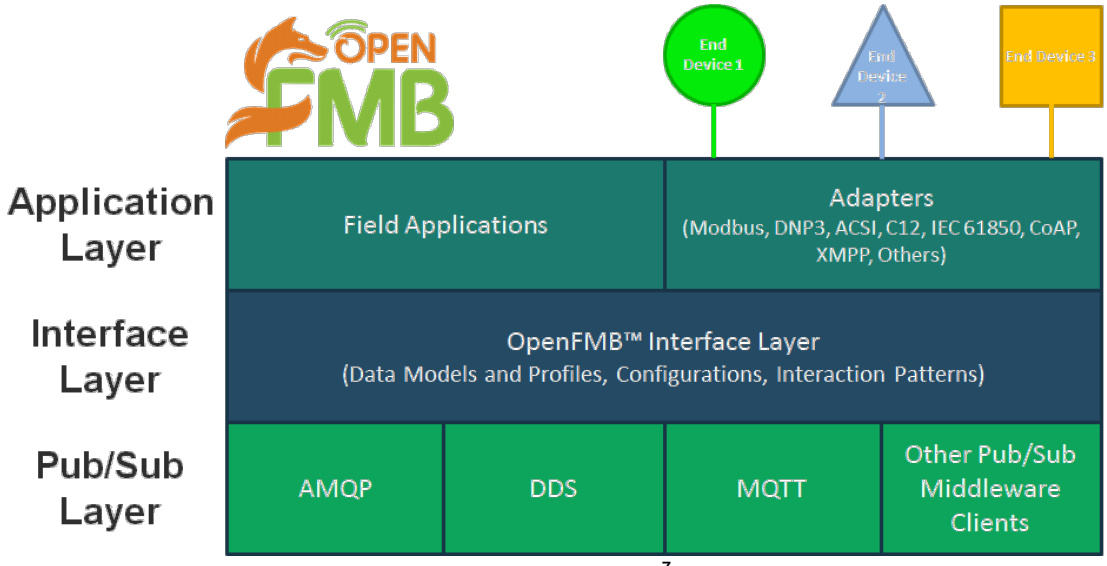
3.2. SGIP

SGIP (Smart Grid Interoperability Panel) is the key player addressing the interoperability of various devices connected to the grid system. “SGIP does not directly develop or write Smart Grid standards but instead brings together various, invested stakeholders to identify the requirements and accelerate the development of "interoperable standards." [6]

The OpenFMB (Open Field Message Bus), an SGIP initiative, is a message bus protocol that supports “Distributed Energy Resource Management”. It defines a framework for distributed intelligent nodes to interact with each other through loosely coupled, peer -to-peer messaging.[6]

The OpenFMB is gaining traction in the industry because of its “scalable peer-to-peer publish/subscribe architecture” [6] and “Data-centric rather than device-centric communication” support.[6]

3.3. OpenFMB Architecture [7]



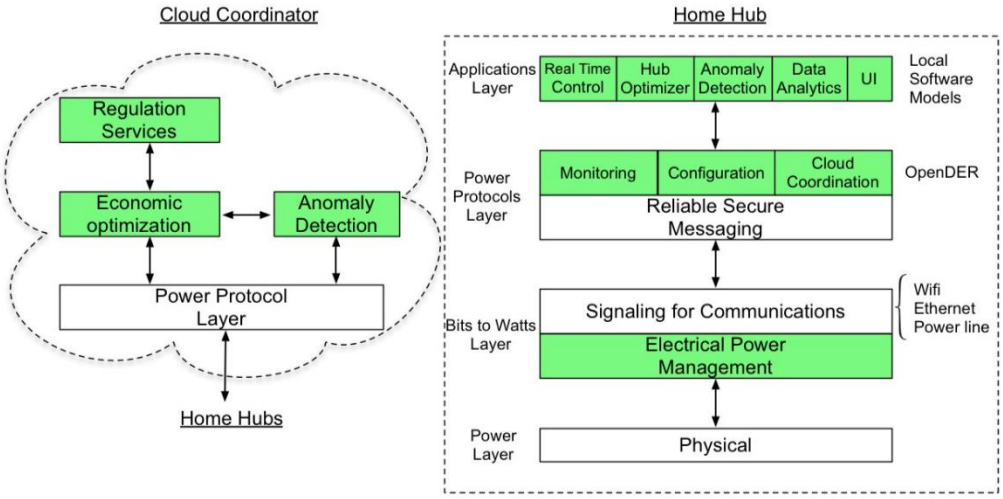
OpenFMB Logical Architecture [7]

The logical architecture for the specification is a layered approach with the following layers:

1. **OpenFMB Pub/Sub layer** [7]– supports multiple protocols such as DDS, AMQP, MQTT and other publish/subscribe middleware clients[7]
2. **OpenFMB Interface layer** [7] – supports multiple information models and profiles, configurations, and interaction patterns [7]
3. **Application layer** [7]– supports field applications and equipment/protocol-specific adapters for protocols such as Modbus, IEC 61850, DNP3, C12, CoAP, XMPP, and others[7]

4. System design

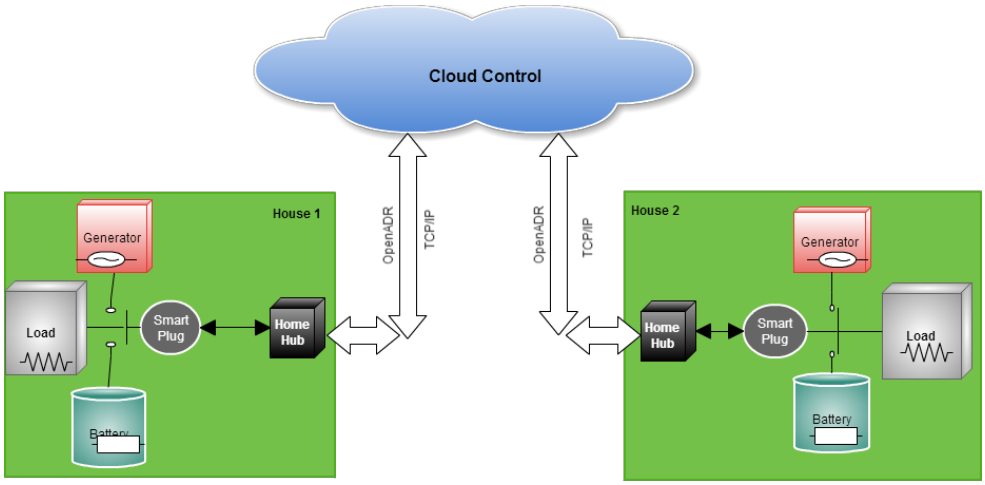
"Powernet" is composed of the "Cloud Control" (CC) and the Home Hub (HH). The Home Hub monitors and controls the local devices based on the power price, and periodically send device status (power consumption) to the "Cloud Control" (CC). "Cloud Control" collects device statuses from "Home Hub", determines the desired net load and optimizes the power flow based on the aggregated data.



Hierarchies of Powernet and their functionalities.

Our project mainly focuses on the "Cloud Control" part, and provides the Power Protocol Layer to Home Hubs. Additionally "Dashboard" is implemented to provide real time power usage by the Home Hub using informative graphs.

4.1. System Overview



System Overview

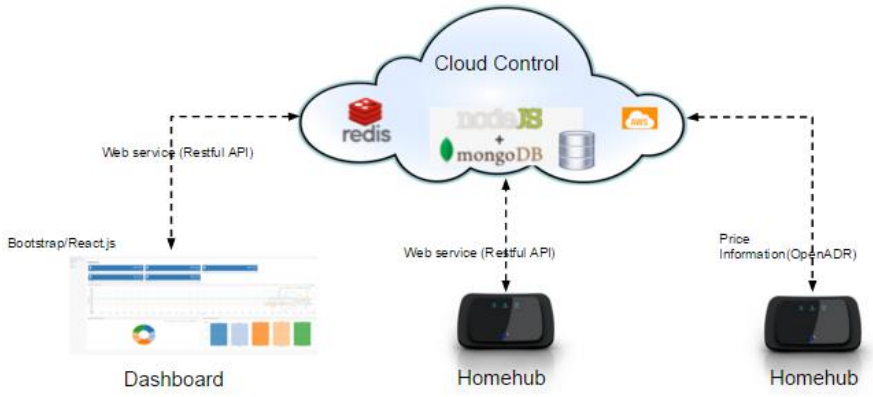
- **Cloud Control**
  The cloud control is the main component that is hosted on the cloud infrastructure (Amazon Web Service) and is intended to interface with the OpenADR for price information. It also provided a host of webservers for "Homehub" and "Dashboard" via RESTful API.

- **Home Hub**

The Homehub represents a component in the house that interacts with the Cloud control over the TCP/IP network and retrieves useful information e.g. “pricing information” from the web service provided by “Cloud Control”. Additionally it sends useful information to the “Cloud Control” for e.g. “total power usage” that could be utilized for future data analytics.

- **Dashboard**
  The dashboard provided power usage by Home Hubs using informative graphs. It directly interacts with the **“Cloud Control”** via RestFul API for real time data.

4.2. Low level Design



System Components

- **Cloud Control**
  The “Cloud Control”, hosted on AWS(Amazon Web Service) is implemented using node.js and backend database “mongoDB”. An in memory cache ”redis” is used to improve the performance of data exchange between the dash board and the “Cloud Control”
- **Dashboard**
  The dashboard is implemented using the “bootstrap” and “React-NVD3” library. “React-NVD3” provides predefined components such as maps, charts and line graph which can be incorporated to present rich UI. Additionally, “Leaflet” library is used to provide/represent location information.
- **Homehub**
  This is being implemented by the OpenBMS team.

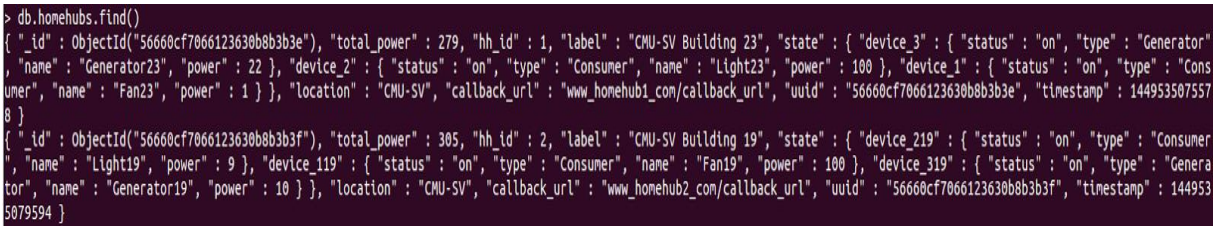
5. System implementation

Our “Cloud Control” architecture comprises of three different layers – persistent layer, service layer, view layer.

Persistent Layer

MongoDB is used as the backend persistent server. There are two collections (tables) in MongoDB - homehubs and hhstatus.

homehubs is used to store the basic configuration of each Home Hub. The schema design is shown in below.



```
> db.homehubs.find()
{ "_id" : ObjectId("56660cf7066123630b8b3b3e"), "total_power" : 279, "hh_id" : 1, "label" : "CMU-SV Building 23", "state" : { "device_3" : { "status" : "on", "type" : "Generator"
, "name" : "Generator23", "power" : 22 }, "device_2" : { "status" : "on", "type" : "Consumer", "name" : "Light23", "power" : 100 }, "device_1" : { "status" : "on", "type" : "Cons
umer", "name" : "Fan23", "power" : 1 } }, "location" : "CMU-SV", "callback_url" : "www_homehub1_com/callback_url", "uuid" : "56660cf7066123630b8b3b3e", "timestamp" : 144953507557
8 }
{ "_id" : ObjectId("56660cf7066123630b8b3b3f"), "total_power" : 305, "hh_id" : 2, "label" : "CMU-SV Building 19", "state" : { "device_219" : { "status" : "on", "type" : "Consumer
", "name" : "Light19", "power" : 9 }, "device_119" : { "status" : "on", "type" : "Consumer", "name" : "Fan19", "power" : 100 }, "device_319" : { "status" : "on", "type" : "Genera
tor", "name" : "Generator19", "power" : 10 } }, "location" : "CMU-SV", "callback_url" : "www_homehub2_com/callback_url", "uuid" : "56660cf7066123630b8b3b3f", "timestamp" : 144953
5079594 }
```

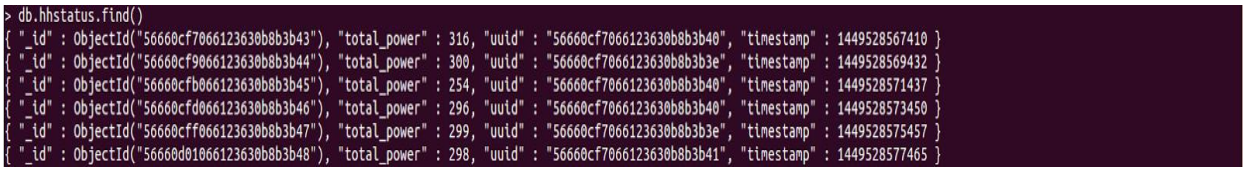

The homehubs schema design

Each field is explained as follows:

1. **_id** - The unique id of each record in homehubs.

2. **total_power** - Current aggregated power consumption of each device connected to the Home Hub.
3. **label** - The name of the Home Hub.
4. **state** - The device list in the connected to the Home Hub, which includes the unique device id, the device name, the device type, the device status and its power consumption.
5. **local** - The geolocation of the Home Hub.
6. **callback_url** - The callback_url is used to send the price change information or other control command back to Home Hub.
7. **uuid** - The Universally Unique IDentifier of the Home Hub
8. **timestamp** - The epoch time when the record is last updated.

hhstatus is used to store the status histories of each Home Hub. Once there is a status update for an HH, a new record is appended to this collection. The schema design is shown below



```
> db.hhstatus.find()
{ "_id" : ObjectId("56660cf7066123630b8b3b43"), "total_power" : 316, "uuid" : "56660cf7066123630b8b3b40", "timestamp" : 1449528567410 }
{ "_id" : ObjectId("56660cf9066123630b8b3b44"), "total_power" : 300, "uuid" : "56660cf7066123630b8b3b3e", "timestamp" : 1449528569432 }
{ "_id" : ObjectId("56660cfb066123630b8b3b45"), "total_power" : 254, "uuid" : "56660cf7066123630b8b3b40", "timestamp" : 1449528571437 }
{ "_id" : ObjectId("56660cfd066123630b8b3b46"), "total_power" : 296, "uuid" : "56660cf7066123630b8b3b40", "timestamp" : 1449528573450 }
{ "_id" : ObjectId("56660cff066123630b8b3b47"), "total_power" : 299, "uuid" : "56660cf7066123630b8b3b3e", "timestamp" : 1449528575457 }
{ "_id" : ObjectId("56660d0106123630b8b3b48"), "total_power" : 298, "uuid" : "56660cf7066123630b8b3b41", "timestamp" : 1449528577465 }
```


The hhstatus schema design

Each field is explained below:

1. **_id -** The unique id of each record in hhstatus
2. **total_power** - The aggregated power consumption of each device connected to the Home Hub
3. **uuid** - The Universally Unique IDentifier of the Home Hub
4. **timestamp** - The epoch time when HH sends the status update request to the CC

Service Layer

A Node.js+Express server is used to host the RESTful APIs and dashboard. The central routine is setup up through the app.js file. This program run's the web service, connects to the database, and sets the appropriate routes for the RESTful APIs and dashboard AJAX queries.

The RESTful APIs fall into three different categories

1. API for OpenBMS,
2. API for OpenADR and
3. API for dashboard.

#	REST APIs	Method	Usage
1	/api/homehubs/	POST	OpenBMS
2	/api/homehubs/<uuid>/	PATCH	OpenBMS
3	/api/price/	POST	OpenADR
4	/api/homehubs/aggregation/<timestamp>	GET	Dashboard
5	/api/homehubs/	GET	Dashboard

RESTful APIs

1. **/api/homehubs (POST)**: Used by OpenBMS. When there is a new Home Hub, OpenBMS will call this API to register the new HH with the "Cloud Control".
   Parameters - JSON formatted post request body

```
{
        "uuid" : Universally Unique ID for HH,
        "label": User configured label for HH,
        "total_power": aggregate power consumption at HH,
        "location": location information,
        "callback_url": URL to post data from CC to HH,
        "state":{
                device id: {
                        "power" : device power consumption,
                        "status"   : device status,
                        …
                },
                …
        }
}
```

   Response - Status code 201 with {"uuid" : uuid of the newly registered Home Hub}

2. **/api/homehubs/<uuid>/ (PATCH)** - Used by OpenBMS. When there is status change in Home Hub, OpenBMS will call this API to send the updated status data to the "Cloud Control".
   Parameters - JSON formatted post request body

```
{
    "total_power" : consumption
}
```

   Response - Status code 200

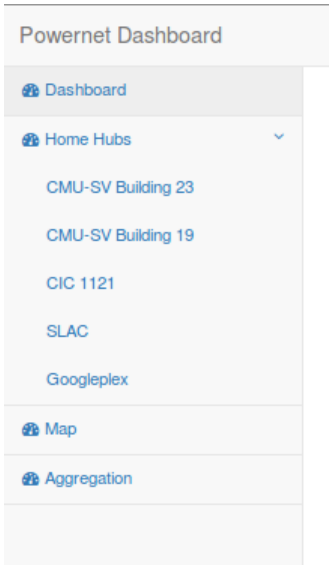
3. **/api/price/ (POST)** - Used by OpenADR. Once there is a price change, OpenADR will send the changed power price with API.
   Parameters -
   {
   "price" : latest power price
   }
   Response - Status code 200.

4. **/api/homehubs/aggregation/<timestamp>/ (GET)** - Used by Dashboard. It returns all the history data of each Home Hub during a specified time period.
   Parameters - None
   [
   { "key": Home Hub Label 1,
   "values": [
   [timestamp1, power_consumption1], [timestamp2, power_consumption2], ...]
   },
   { "key": Home Hub Label 2,
   "values": [
   [timestamp1, power_consumption1], [timestamp2, power_consumption2], ...]
   }
   ]
   Response - Status code 200 with following JSON response string

5. **/api/homehubs/ (GET)** - Used by Dashboard. It returns the current status of each Home Hub.
   Parameters - Node
   [
   {
   "hh_id": Home Hub uuid 1,
   "name": Home Hub Label 1,
   "total_power": consumption 1
   },
   {
   "hh_id": Home Hub uuid 2,
   "name": Home Hub Label 2,
   "total_power": consumption 2
   },
   ]
   Response - Status code 200 with following JSON response string

View Layer

The view part of the dashboard model is comprised of React components. All pages share the same navigation component for the dashboard effect. The project supports partial isomorphism (server/client-side DOM/JS equality) and uses jQuery for real-time AJAX requests.



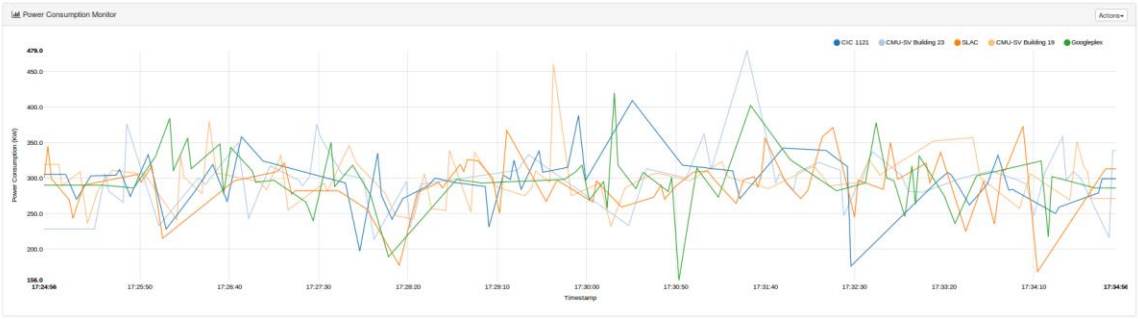
Navigation Bar

Currently, the pages supported are:

1. **Dashboard Index [single instance]**

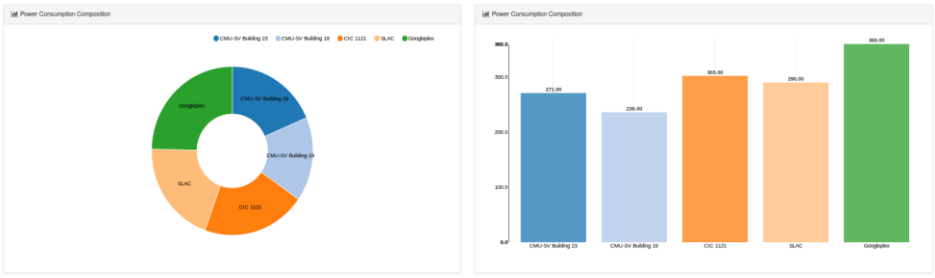
This view page aggregates the data from each home Hub and displays the data in a master panel. At the moment, this panel comprises of the following components:

- ❏ Power Consumption Line Chart



Histogram Power Consumption Line Chart

- ❏ Power Consumption Donut Chart
- ❏ Power Consumption Bar Chart



Dynamic Power Consumption Donut and Bar Charts

These components all are different ways to displaying the data. Although the data supplied to each component is the same, each can be tweaked to include other information in real-time.

To model the data, d3 in the form of nvd3 is used for real-time line, bar, and donut graphs. Additionally, JSON payloads (listed above in Service Layer) are fed into the home Hub components.

2. **Individual Home Hub Content [dynamic instance]**

The view page for a particular “Home Hub” consists of a table that will contain information desired by the operator. In this instance, the information displayed is the list of devices under the “Home Hub” control. Additionally, more components may be added to increase the analysis at a more granular level.

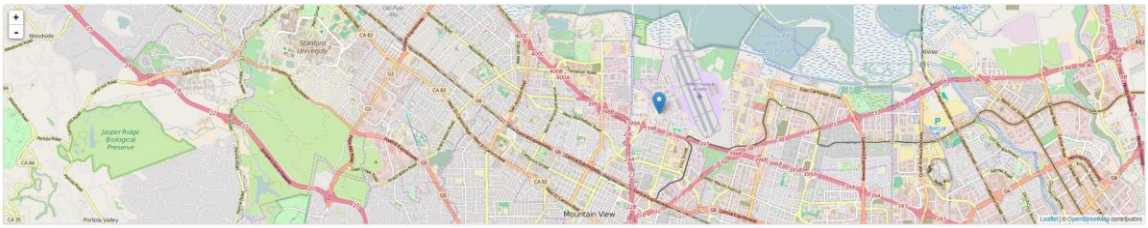
Context classNames

Device Name	Device Type	Status
Champion Power Equipment Model 41537	Generator	on
Super awesome fan	Load	on
Champion Power Equipment Model 41537	Generator	on
Champion Power Equipment Model 41537	Storage	off
Champion Power Equipment Model 41537	Generator	on

Home Hub Specified Content

3. **Map [single instance]**

The map view page will contain a dynamic list of home hub markers using the GPS coordinates of each deployment. Clicking on “marker” will popup information about the specified “Home Hub”.



Home Hub Map centered on CMU-SV

A main navigation component is used to create the main dashboard layout. This component is the root component comprised of a navigation bar and main view sub-components.

6. Experiments and analysis

To finalize the frameworks to use, we compared a list of systems (mainly for the web server and database). We found two major web frameworks namely Node.js and Django.

Node.js

It is a rapidly growing technology. It is an asynchronous event driven framework, and is designed to build scalable network applications.

Advantages:

1. It is designed for coordination with mongoDB
2. It is powered with react for efficient frontend rendering and re-rendering. (Critical to a real time web application like Dashboard)

Disadvantages:

1. Learning curve for Node.js, JavaScript, and React
2. It is hard to debug

Django

It is a high-level Python Web framework that encourages rapid development and clean, pragmatic design.

Advantages :

1. Fast - Django was designed to help developers take applications from concept to completion as quickly as possible.
2. Secure - Django takes security seriously and helps developers avoid many common security mistakes

Disadvantages:

1. It is not designed for coordination with MongoDB.
2. It works well with traditional Relational Database Systems (MySQL, PostgreSQL), however the latest Django does not support MongoDB anymore, and that is the main reason that why we choose Node.js as our backend web server.

MongoDB

We also compared MongoDB (which is a NoSQL database) with traditional Relational Database Systems. MongoDB databases provide a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases. In addition to its scalability and superior performance

Advantages:

1. Large volumes of rapidly changing structured, semi-structured, and unstructured data
2. JSON based collection, which is easy for manipulation
3. Geographically distributed scale-out architecture instead of expensive, monolithic architecture

Disadvantages:

1. It does not support bulk transactions and the join operation.
2. Does not have a good transactional support

Traditional Relational Database Systems (RDBMs)

We found that they have good support for transactions and join operations. However, their schemas are inflexible, and they are not designed for scalability.

Conclusion

As our specific requirement did not need a constant updates and only append operation was required together with high scalability , we decided in favor of MongoDB as its inherent limitation, transactional support limitation, does not pose an major issue for us.

7. Conclusions /future work/Known Issue

The basic function of the Cloud Control works fine, and it meets the requirement for real-time processing. The response time is less than one second. Although the groundwork has been laid out, much more work needs to be done to reach production-grade material. In particular,

Integration

1. Integration testing with OpenBMS and a large amount of Home Hubs
    a. Here, actual data should be used to resemble actual use
    b. Trial and error will help see how the system scales, what is needed, and what is not needed
2. Integration of OpenADR functionality into the Cloud Controller and OpenDER/OpenBMS communication
    a. Cloud Controller can operate as a VEN and forward market context pricing signals to the subscribed Home Hubs

Security

1. Manage bottlenecks to avoid DDoS attacks
2. Run a vulnerability assessment and penetration test
3. Obtain a certificate in order to use HTTPS (confidentiality and authenticity)

Performance

1. Implement Redis cache layer and periodically back up the non-volatile hard drive.
    a. Can be sync'd with the Home Hubs to only carry fresh data that can be restored in the case of an non-conditional termination

The **Powernet/OpenDER** project has built the foundational cloud infrastructure for the specified position in bring information technology to the electrical systems' operational technology. In addition to a RESTful API communication with "Home Hubs", a "Dashboard" to pick-and-choose the data has been built.

8. Reference

1. Intern Guidelines: Practicum proposal
2. "The Grid." *GC1MFTY (Traditional Cache) in Oklahoma, United States Created by 3PawCaching*. Web. 11 Dec. 2015.
3. "MITSUBISHI ELECTRIC Environment - Environmental Report - Smart Grid Demonstration Project." *MITSUBISHI ELECTRIC Environment - Environmental Report - Smart Grid Demonstration Project*. Mitsubishi Electric, n.d. Web. 06 Dec. 2015. <http://www.mitsubishielectric.com/company/environment/report/products/randd/smartgrid>.
4. "Smart Grid." *Smart Grid*. Web. 11 Dec. 2015. <http://energy.gov/oe/services/technology-development/smart-grid>.
5. "Frequently Asked Questions." *FAQ*. Web. 11 Dec. 2015. <http://www.openadr.org/faq>.
6. "SGIP FAQs." *SGIP*. Web. 11 Dec. 2015. <http://www.sgip.org/FAQs>.
7. "Open Field Message Bus (OpenFMB™) Project." *SGIP*. Web. 11 Dec. 2015. <http://sgip.org/Open-Field-Message-Bus-OpenFMB-Project>.

9. Other References

1. "Control Your Energy Future." *Home*. N.p., n.d. Web. 06 Dec. 2015. <http://www.openadr.org/>.
2. "Brand Guidelines." *SGIP*. N.p., n.d. Web. 06 Dec. 2015. <http://sgip.org/Brand-Guidelines>.
3. "Node.js." *Node.js*. N.p., n.d. Web. 06 Dec. 2015. <https://nodejs.org/en/>.
4. "Launch Your GIANT Idea." *MongoDB for GIANT Ideas*. N.p., n.d. Web. 06 Dec. 2015. <https://www.mongodb.org/>.
5. "Amazon Web Services (AWS) - Cloud Computing Services." *Amazon Web Services, Inc.* N.p., n.d. Web. 06 Dec. 2015. <https://aws.amazon.com/>.
6. "*Documentation." *Redis*. N.p., n.d. Web. 06 Dec. 2015. <http://redis.io/documentation>.
7. Chatzivasileiadis, Spyros, et al. "Cyber physical modeling of distributed resources for distribution system operations." *arXiv preprint arXiv:1505.00078* (2015).
8. Laval, Stuart; Godwin, Bill. “Distributed Intelligence Platform (DIP) Reference Architecture: Volume 1.” Duke Energy Corporation. 2015.
9. McMillin, Bruce. “Use Case: Distributed Power Balancing in openFMB.” Missouri University of Science and Technology, Department of Computer Science. 2015.
10. SGIP OpenFMB Working Group. “Open Field Message (OpenFMB) Bus Project.” OpenFMB Priority Action Plan. 2015.

10. Appendix: Software Information

10.1. GitHub Account

The following github account is used for the project
https://github.com/Cpruce/OpenDER

10.2. Installation Procedure

- **Cloning**:
  Clone the git hub repository (OpenDER and bootstrap_dashboard) using the following command
  git clone https://github.com/Cpruce/OpenDER.git

git clone https://github.com/Cpruce/bootstrap_dashboard.git

- **Dependency Resolution :**
  Resolve dependency using the npm package
  npm install
  Note: In case of error run “npm install” command after deleting the folder node_modules using the command
  rm –rf node_modules

- **Build:**
  In order to compile and bundle the necessary files for the frontend and backend packages, a combination of webpack and gulp is used, especially to convert any jsx (html/css in javascript) files to js, html, and css files. This build configuration resides in the file aptly named ‘gulpfile’.

  The gulp configuration for this project specifies the follow build actions:
  gulp frontend-build - bundles the frontend components in /public/ and translates using the babel-loader for jsx transformations
  gulp backend-build - bundles the routes
  gulp frontend-watch - runs with the argument frontend-build with callback signaling completion
  gulp backend-watch - runs with the argument backend-build with callback signaling completion and a server restart for synchronization
  gulp run - runs frontend-watch, backend-watch, and the node server

- **Running the server:**
  gulp run - runs frontend-watch, backend-watch, and the node server

If the server runs without any error the command prompt displays the port on which the server is listening. Open the browser and connect to the displayed URL

10.3. Contact Information

Sl. No	Name	Contact No	Mail ID
1	Praveen Gandala	4088345482	Pravk.reddy@gmail.com
2	Cory Pruce	8622687092	corypruce@gmail.com
3	Bixian Bao	6502379850	skipper.bao@gmail.com
4	Yizhen Chen	NA	yzchen1991@gmail.com

10.4. Contribution & Responsibility

Sl. No	Area	Owners
1	Front End	Yizhen, Praveen, Cory
2	Back End	Bixian, Praveen
3	OpenADR Porting	Yizhen, Bixian, Cory