Spring 7-18-2018

Adoption of an Internet of Things Framework for Distributed Energy Resource Coordination and Control

Tylor Slay
Portland State University

Adoption of an Internet of Things Framework for Distributed Energy Resource Coordination and Control

by

Tylor Slay

A thesis submitted in partial fulfillment of the requirements
for the degree of

Master of Science
in
Electrical and Computer Engineering

Thesis Committee:
Robert Bass, Chair
Jonathan Bird
Mark Faust

Portland State University
2018

Abstract

Increasing penetration of non-dispatchable renewable energy resources and greater peak power demand present growing challenges to Bulk Power System (BPS) reliability and resilience. This research investigates the use of an Internet of Things (IoT) framework for large scale Distributed Energy Resource (DER) aggregation and control to reduce energy imbalance caused by stochastic renewable generation. The aggregator developed for this research is Distributed Energy Resource Aggregation System (DERAS).

DERAS comprises two AllJoyn applications written in C++. The first application is the Energy Management System (EMS), which aggregates, emulates, and controls connected DERs. The second application is the Distributed Management System (DMS), which is the interface between AllJoyn and the physical DER.

The EMS runs on a cloud-based server with an allocated 8 GB of memory and an 8 thread, 2 GHz processor. Raspberry Pis host the simulated Battery Energy Storage System (BESS) or electric water heater (EWH) DMSs. Five Raspberry Pis were used to simulate 250 DMSs.

The EMS used PJM's regulation control signals, RegA and RegD, to determine DERAS performance metrics. PJM is a regional transmission organization (RTO). Their regulation control signals direct power resources to negate load and generation imbalances within the BPS.

DERAS's performance was measured by the EMS server resource usage, network data transfer, and signal delay. The regulation capability of aggregated DER was measured using PJM's resource performance assessment criteria. We found the use of an IoT framework for DER aggregation and control to be inadequate in the current network implementation. However, the emulated modes and aggregation response to the regulated control signal demonstrates an excellent opportunity for DER to benefit the BPS.

Dedication

To the endless pursuit of knowledge.

Acknowledgements

I would like to acknowledge Michael Davis, Annie Clarke, Leighton Clarke, and Crystal Eppinger for their help and support with this research. A special thanks goes to Linda Rankin for her expert guidance and moral support.

Finally, I would like to put a spotlight on Dr. Robert Bass who has facilitated such an amazing research environment. A true mentor allows their mentee to learn lessons on their own, offering guidance and support when the time is right. Thank you for allowing me to grow as a researcher.

Contents

List of Tables

List of Figures

Acronyms

BESS Battery Energy Storage System

BESSs Battery Energy Storage Systems

BPS Bulk Power System

CPU Central Processing Unit

DAQ data acquisition

DER Distributed Energy Resource

DERAS Distributed Energy Resource Aggregation System

DERs Distributed Energy Resources

DMS Distributed Management System

DMSs Distributed Management Systems

DOE U.S. Department of Energy

DSG Dispatchable Standby Generation

EMS Energy Management System

EV Electric Vehicle

EVs Electric Vehicles

EWH electric water heater

EWHs electric water heaters

GPIO general-purpose input/output

HVAC Heating Ventilation and Air Conditioning

IoT Internet of Things

LAN Local Area Network

NERC North American Electric Reliability Corporation

OCF Open Connectivity Foundation

OS Operating System

PCC Point of Common Coupling

PGE Portland General Electric

PNNL Pacific Northwest National Laboratory

PSU Portland State University

PV Photovoltaic

RDSI Renewable and Distributed Systems Integration Program

RPS Renewable Portfolio Standard

RTO regional transmission organization

SGDP Smart Grid Demonstration Program

SGIG Smart Grid Investment Grant Program

SoC State of Charge

UML Unified Modeling Language

VPN Virtual Private Network

WAN Wide Area Network

1 Introduction

1.1 Problem Statement

Increased penetration of non-dispatchable renewable energy resources and greater peak power demand present growing challenges to the bulk power system's reliability and resiliency. The stochastic generation profiles of resources like wind and solar can lead to large imbalances between generation and load, which increase the required reserve margin in the balancing area. This research proposes an open-source communication platform to utilize readily-available DER to balance the stochastic generation of renewable energy resources and reduce peak demand.

1.2 Motivation

In 2016, renewable energy accounted for nearly two-thirds of added generation capacity worldwide, totaling 165 gigawatts of new generation resources [3]. Figure 1.1 predicts Denmark will use renewable resources to generate nearly 70% of its electricity by 2022, with Ireland, Spain, Germany, and the United Kingdom around 25%. Renewable energy resources are no longer an enthusiast's fad. Wind and solar generation are becoming cost competitive with fossil fuels and have the added benefit of a free and indefinite fuel supply.

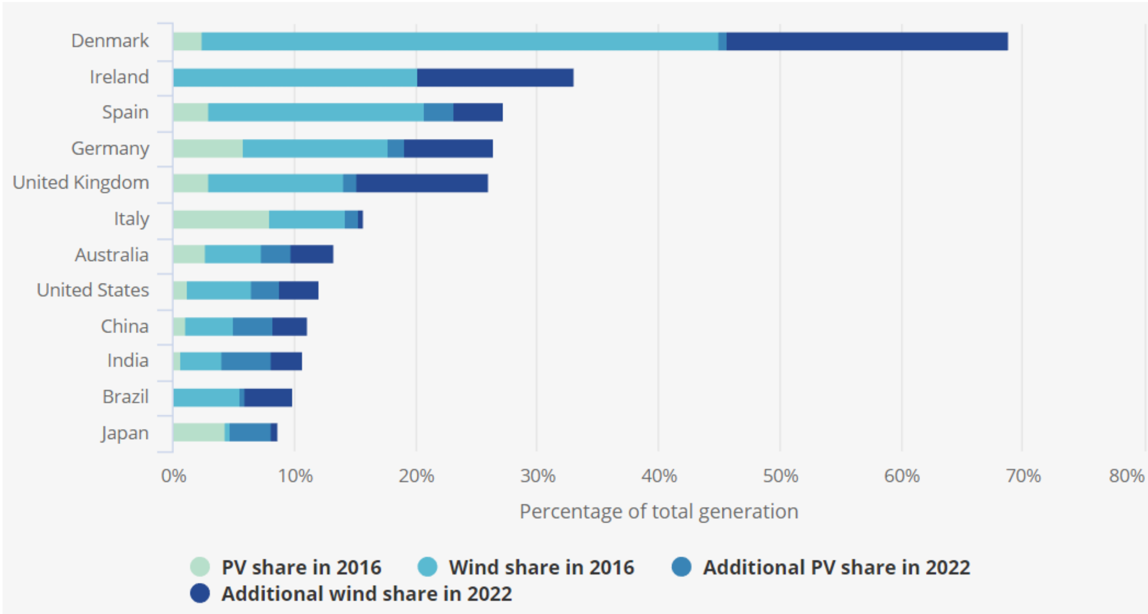

Figure 1.1: Forecast of annual electricity generation for select countries 2016-2022 [2]

As a whole, the United States may only exceed 10% renewable generation by 2022, but many individual states are pushing for 25-50% renewable generation by 2040, as detailed in Figure 1.2. Hawaii is an outlier, aiming for 100% renewable generation by 2045. Hawaii is the perfect candidate for renewable generation because imported oil accounted for 70% of its energy generation in 2015 [14]. The cost to import the oil required to power Hawaii makes electricity prices there the most expensive in the United States, at an average of $0.25 per kilowatt-hour. For comparison, the average price for electricity for the rest of the United States is $0.10 [14]. Due to cheap fossil fuel generation within the United States, many states have not adopted Renewable Portfolio Standard (RPS) goals. The non-dispatchable and stochastic nature of wind and solar generation has also slowed adoption.

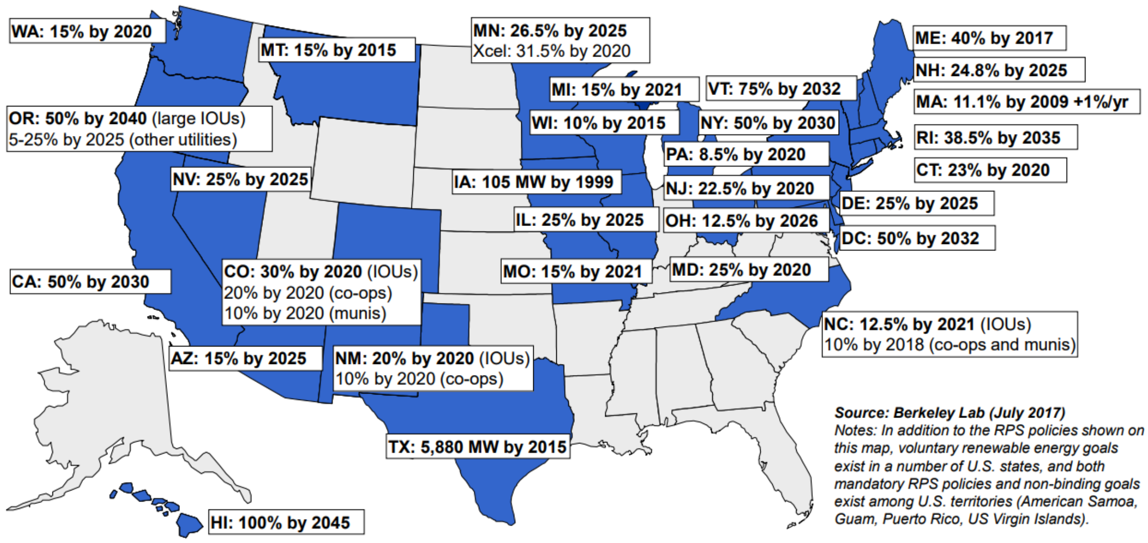

Figure 1.2: United States Renewable Portfolio Standards [3]

Figures 1.3 and 1.4 demonstrate how the stochastic nature of wind and solar can disrupt the constant effort to balance generation and load within the BPS. The left plot within Figure 1.3 shows the effect of wind and solar during optimal conditions assuming solar and wind generation are consistent and predictable throughout the day. The overall affects of wind and solar generation are telegraphed at the peaks of each day. Each day transitions from low to high with very little volatility. In the right plot within Figure 1.3, there are multiple spikes within peak hours due to changes in wind and solar production.

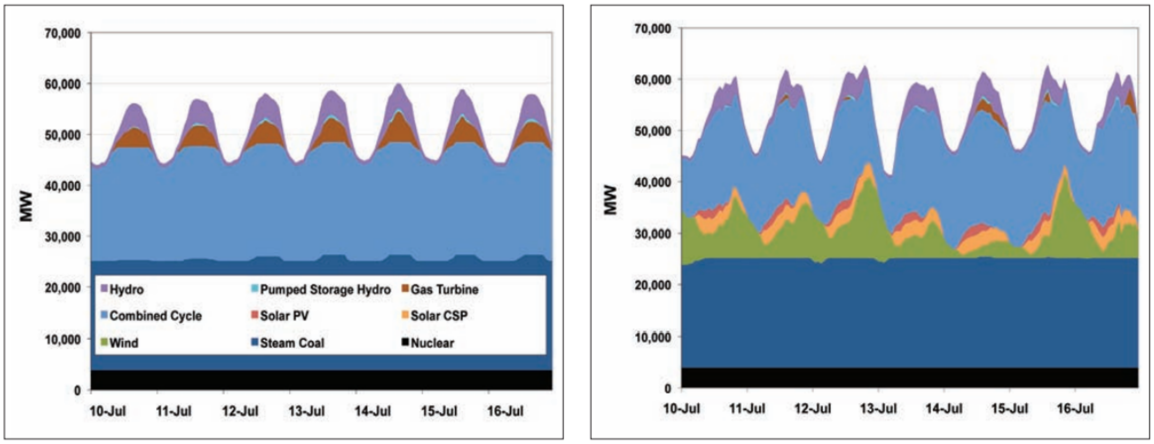

Figure 1.3: Impact of 35% Renewables generation under optimal conditions [4]

Figure 1.4 shows a worst-case scenario for wind and solar generation. There are large deviations in both the magnitude of generation to meet load and the ability to sustain generation for the duration of peak load. Large imbalances in load and generation lead to swings in frequency and voltage that impinge on strict regulatory requirements. The bulk power system must acquire a greater margin of reserve generation to counteract the imbalance caused by the high penetration of renewable energy generation.

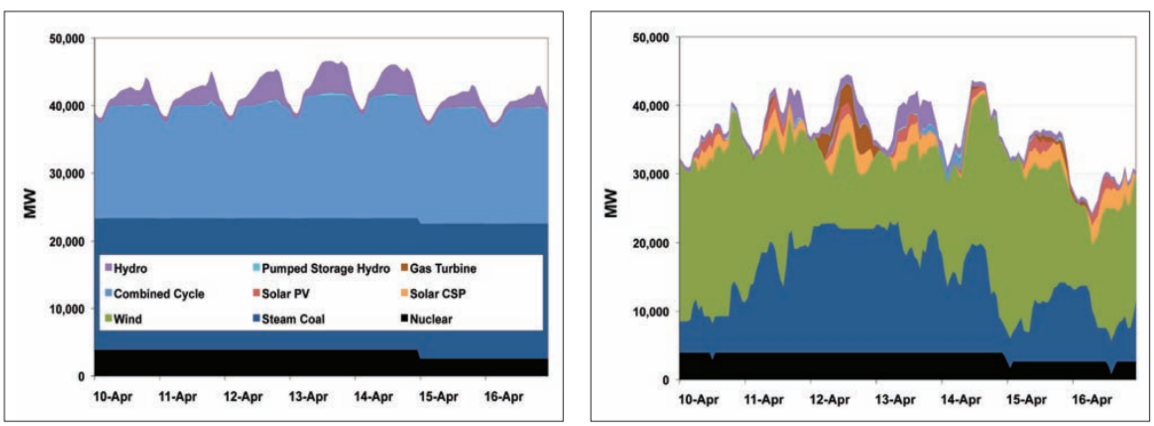

Figure 1.4: Impact of 35% Renewables generation under worst case conditions [4]

Figure 1.5 shows the projected increase in the required reserve margin for the Western

Electricity Coordinating Council (WECC) balancing region using probabilistic forecasts. Wind and solar generation are not the only cause for the increased reserve requirements, but they are two of the dominating factors. Increasing the reserve generation in a balancing region increases customer billing rates because these additional resources do not generate revenue. Reserve generators typically cycle on/off far more often than baseload generators, thereby increasing costs. They also often operate at reduced efficiency because they are called upon to operate at non-optimal setpoints.

Figure 1.5: NERC - WECC reserve margin summary [5]

2 Literature Summary

2.1 Literature Summary

The U.S. Department of Energy (DOE) has spent over $4 billion funding three smart grid programs: Renewable and Distributed Systems Integration Program (RDSI), Smart Grid Demonstration Program (SGDP), and Smart Grid Investment Grant Program (SGIG), totaling 140 projects [15]. The objectives of this research align with the RDSI program, which awarded only nine projects in 2008. None of the currently awarded projects proposed a device-agnostic, open-source communication platform. Portland General Electric (PGE) is currently supervising several pilot projects focused on renewable and distributed systems integration, including the research presented in this paper [4].

2.1.1 GenOnSys

PGE currently operates a Dispatchable Standby Generation (DSG) program called GenOnSys, which aggregates standby generators to provide non-spinning generation reserves [16]. GenOnSys comprises four primary parts: a communication backbone, asset controllers, intelligent metering, and a centralized control center. GenOnSys was developed to be a flexible system for dispatching all forms of distributed generation, but was not designed to scale with the adoption of hundreds of thousands of devices. This was the primary reason PGE has funded the research presented in this thesis.

2.1.2 VOLTTRON

The leading utility-centered distributed control platform is VOLTTRON, developed by the DOE at Pacific Northwest National Laboratory (PNNL). Figure 2.1 shows the platform user interface for VOLTTRON agents. VOLTTRON provides utilities with the means to manage different assets within the power system at a large scale. VOLTTRON has the ability to communicate with thousands of "agents", applications running on a device, with only a small round-trip signal delay [6]. VOLTTRON uses the ZeroMQ message bus for its built-in security and fast communication. Control signals can be translated to MODBUS, BACnet, DNP3, and SEP 2.0 device control signals, thereby providing interoperablility with a wide range of utility assets. VOLTTRON is by far the most developed and well-rounded distributed control platform available for communication and control of power system assets. However, it was not designed as an IoT framework for consumer devices such as Electric Vehicles (EVs), Heating Ventilation and Air Conditioning (HVAC) systems, and electric water heaters (EWHs).

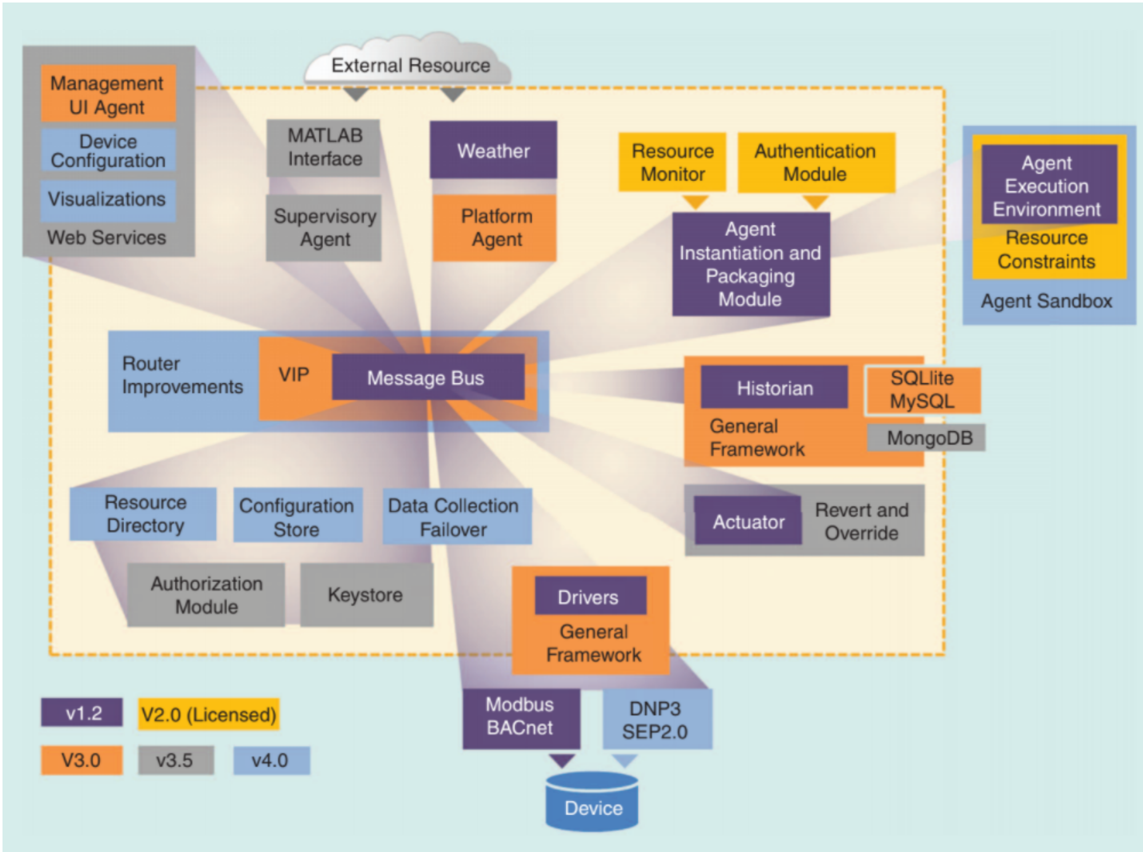

Figure 2.1: VOLTTRON platform user interface [6]

2.1.3 IoTivity

IoTivity, sponsored by the Open Connectivity Foundation (OCF), is an IoT framework designed around the resource-based RESTful architecture model. It supports development in several languages and operating systems, and operates as a bridge between other connectivity platforms. Figure 2.2 shows how the framework supports a variety of transport protocols, while creating consumer profiles to drive interoperability. However, IoTivity's limited consumer market support made it a less desirable candidate for the communication framework for DERAS.

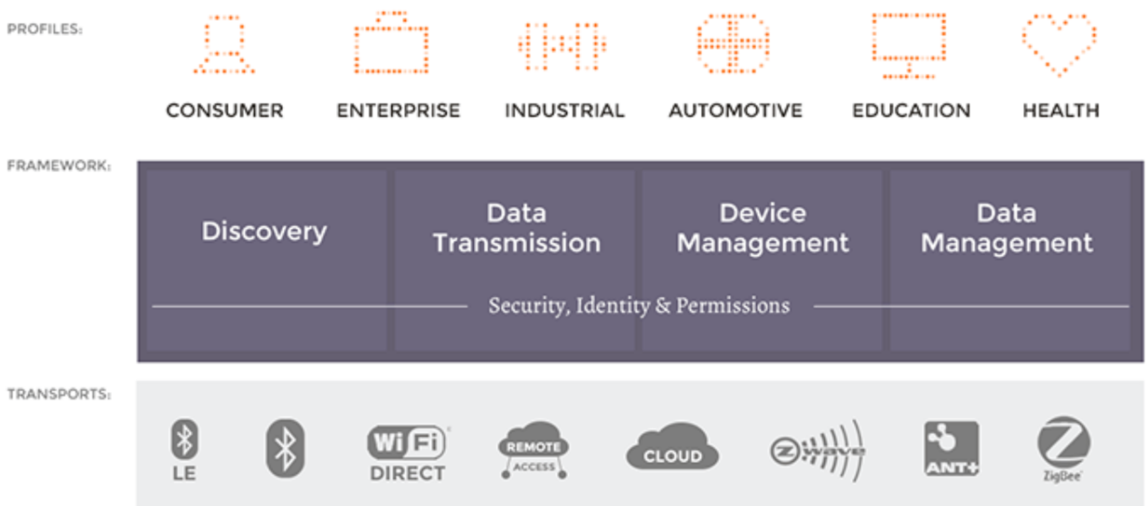

Figure 2.2: IoTivity Framework [7]

2.1.4 OpenThread

Google's OpenThread is the IoT framework behind the widely popular NEST thermostat. OpenThread was designed to be portable and flexible [17]. OpenThread can also be hard-coded to a Central Processing Unit (CPU) or run on top of an operating system, making it architecture-agnostic. It uses the IPv6-based networking protocol, compliant with IEEE 802.15.4-2006 wireless mesh network specification. OpenThread is independent of ZigBee, Z-Wave, and Bluetooth LE, which are established 802.15.4 protocols. Deciding to create another protocol makes OpenThread appear to want to be the default development framework for smart devices. This is dramatically different than IoTivity, which appears to aim to be a bridge between all smart devices.

OpenThread's primary features includes simplicity, security, reliability, efficiency, and scalability. It also supports over 500 end-device connections per single Thread network.

OpenThread's incompatibility with other communication protocols makes it less adaptable for future implementations.

2.1.5 AllJoyn

AllJoyn, developed by the Allseen Alliance, was the final IoT framework adoptred for this research. AllJoyn's framework supports the following features [8]:

- Open source code-base gives developers the ability to inspect the entire AllJoyn framework and contribute in their own way.
- Operating System (OS) independent (i.e., Linux, Windows, Apple, Android, etc).
- Language independent (i.e., C++, Java, C#, JavaScript, and Objective-C).
- Physical network and protocol independence (i.e., Wifi, Bluetooth, Ethernet, etc).
- Service advertisement and discovery that simplifies the process of locating devices and utilizing the services they provide.
- Security support to provide application-to-application communication through advanced security models.

The framework has been backed by major consumer brands such as LG, Sharp, Panasonic, Sony, Sears, Cisco, TP-Link, and Microsoft. Microsoft had even integrated the AllJoyn standard client/router into its Windows 10 OS [18]. AllJoyn was chosen as the framework for DERAS due to the combination of features and consumer backers. The fact that AllJoyn was installed on one of the largest operating system platforms in the world made its adoption by consumer devices very promising.

3 Design Methodology

The proposed system, shown in Figure 3.1, is a bi-level control scheme that facilitates aggregation and modeling of Distributed Energy Resources (DERs). The EMS and DMS, represent the respective priorities of each of the control schemes. The EMS resides within a cloud server and aggregates the available energy (Watt-hour), rated power (Watts), and the ramp rates (Watts per second) of all connected DERs available for import/export dispatch signals. The EMS emulates the average dynamic characteristics of all DERs to create a *digital twin*, which serves as a model to be used for telemetry updates and dispatch allocation.

The DMS is the interface for each DER, interpreting the physical devices parameters and translating them into the required properties for the EMS. Each home may contain multiple DERs that can be aggregated by the EMS.

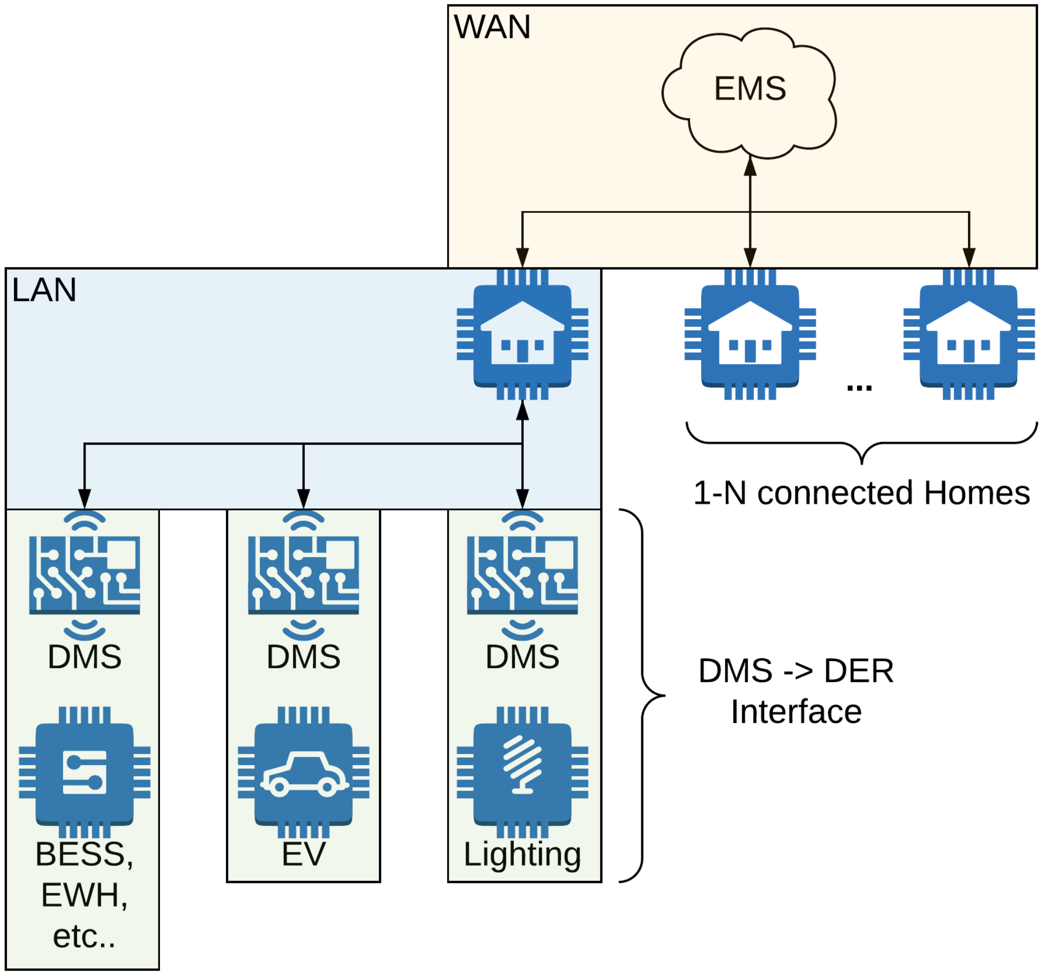

Figure 3.1: DERAS Architecture

The following section summarizes a North American Electric Reliability Corporation (NERC) study of DER adoption effects on the BPS, and how NERC's recommendations influenced the design choices for the proposed system. The AllJoyn Core structure will be introduced along with the rationale for adopting it as the IoT framework for DERAS. PJM's RegA and RegD control signals will be introduced for testing the DERAS network and emulated models, followed by the rationale for the simplified models for both emulation and simulation of DER.

3.1 NERC DER Recommendations

NERC is a not-for-profit international regulatory corporation whose mission is to assure the effective and efficient reduction of risks to the reliability and security of the North American BPS. NERC established a task force in 2015 to develop primary considerations for DER penetration and their affects on reliability. The task force areas of focus are discussed in the following subsections [1].

3.1.1 Modeling

Utilities will no longer be able to bundle DER assets with loads into a single model at the distribution bus for both steady-state and dynamic power system studies. Communication with the DER will be necessary to ensure accurate operations and planning studies, such as:

- Steady-state power flow studies are used to determine real and reactive power flows for the BPS. Power flow calculations can influence network planning, voltage stability, and voltage coordination at the transmission-distribution interface.
- Steady-state short-circuit studies inform equipment short-circuit power levels and highlight voltage sag propagation.
- Dynamic disturbance ride-through studies determine frequency and voltage stability of the BPS following transmission faults.

- Dynamic transient stability studies are used to determine the BPS stability following transmission faults.

3.1.2 Ramping and Variability

The variability of generation resources like wind and solar create problems for BPS coordination, but generally don't cause a considerable change to the overall shape of the daily generation profile. However, ramp rates between morning/evening solar or gusts of wind are new phenomena that dramatically affect the generation/load balance. Aggregated DER can offset these dramatic imbalances.

3.1.3 Reactive Power

Reactive power (VAR), Figure 3.2, represents the resonant energy exchange between capacitive and reactive components within the electric power system. The result is a phase difference between the voltage and current waveforms. This phase shift is represented as the phase angle ϕ, which ranges from 90 to -90 degrees, depending on whether a component is consuming or producing reactive power.

Bus voltages can be increased or decreased through reactive power manipulation. The current interconnection requirements of inverter-based DER systems prevent inverters from providing local voltage support via VAR control. But this is likely to change with increased penetration levels of renewables within the distribution system. It will become imperative to allow these assets to support local voltage levels to maintain BPS reliability.

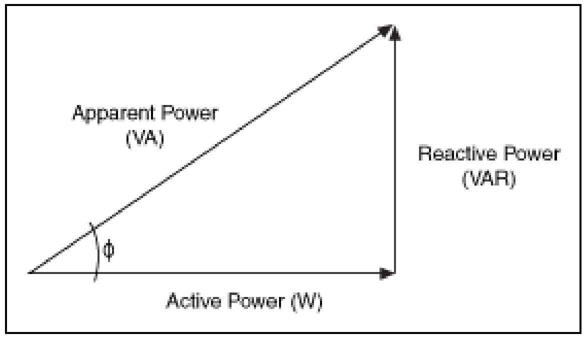

Figure 3.2: Power Triangle

3.1.4 System Protection

At high penetration levels, DERs increase the apparent system impedance causing faults to be more difficult to detect, or cause reverse power flows that require directional relays for fault detection. DER consideration within the BPS will become vital for distribution planning and ensuring reliable and resilient power.

3.1.5 Visibility and Control

BPS operators and coordinators currently have no visibility or control of non-utility scale DER systems. Operations and planning will require real-time information of DERs to maintain a reliable and resilient power system. The data requirements for providing real-time visibility and control to DERs at scale is a challenge that the power industry is trying to overcome. This inspired the choice to use emulated models of DERs to reduce the need for frequent telemetry updates.

3.1.6 Forecasting

Utilities currently use DER forecasts as load modifiers in studies. In a separate report developed by NERC's Integration of Variable Generation Task Force, reliability coordinators use forecasts of aggregate, regional, and individual variable generation [19]. Use of these forecasts improves day-ahead market clearing and improves the system operation reliability and economics. Understanding the uncertainty of variable generation through probabilistic studies also improves system reliability. Specifically, how the chance-of-ramp forecast affects the commitment of generation for reserve margins. Dispatch of DER, such as Electric Vehicle (EV), EWH, and HVAC systems, will also depend on accurate user forecasts to understand the availability of those resources.

3.1.7 Interconnection Requirements

IEEE 1547-2003 [1], is a standard for interconnecting DER with the power grid. The standard covers voltage and frequency ride-through, voltage regulation, re-closing coordination, power quality, and islanding, among other issues. The associated requirements apply to the Point of Common Coupling (PCC) between the grid and the DER. Currently IEEE 1547-2003 prohibits voltage/frequency regulation by DERs at the PCC, but amendment IEEE 1547a would overturn this specification. IEEE 1547-2003 will continue to be revised as DER adoption grows and the potential to affect reliability increases.

Tables 3.1 and 3.2 show the voltage and frequency ride-through specifications for small DERs [1]. As adoption increases, there is a greater potential to lose a significant portion of generation due to a voltage or frequency event.

Voltage Range (pu)	Isolation Time (seconds)
< 0.50	0.16
0.50 < 0.88	2.0
0.88 < 1.10	Run Continuously
1.10 < 1.20	1.0
> 1.20	0.16

Table 3.1: Voltage Ride-Through Conditions for DER sizes < 30 (kW) [1]

Frequency Range (Hz)	Isolation Time (seconds)
> 60.5	0.16
< 59.3	0.16

Table 3.2: Frequency Ride-Through Conditions for DER sizes < 30 (kW) [1]

NERC's task force recommends the following transmission-distributions interface to support each area of focus previously stated [1]:

- DER type (i.e., Photovoltaic (PV), wind, co-generation, etc).
- DER MVA rating.
- Relevant energy production characteristics (i.e., active tracking, fixed tilt, energy storage characteristics, etc.).
- Real and reactive power control functionality.
- DER PCC voltage.
- DER location.
- Date that DER went into operation.

3.2 DERAS Interface

AllJoyn supports two interfaces for automatic advertisement and discovery of applications running within the local network. AllJoyn provider applications, devices that provide a service, broadcast the well-known name of the interface supported by the application to all devices on the network. If there is an AllJoyn consumer application, a device that consumes and/or controls a provided service, it will establish a session with the provider application to maintain visibility and control. AllJoyn's advertisement of an established interface ensures interoperability between provider-consumer applications. We have established the DMS as the provider application and EMS as the consumer application for this research. The AllJoyn interface comprises three primary components: properties, methods, and signals.

Consumer applications use the AllJoyn method call or signal to request information or control a provider application. Figure 3.3 demonstrates the signal exchange for a method call. Once advertisement/discovery is complete, the consumer application can request the provider application to start a process defined by its interface using a method call. The call is sent to the provider application, processed, and then a reply is sent back to the consumer application to confirm the provider application has completed the request.

An AllJoyn signal has two major differences compared to a method call. First, a signal doesn't require an established session for communication. A session can simply broadcast information to any device listening on the network. Second, a signal from a consumer application doesn't require acknowledgment of the signal by the provider application.

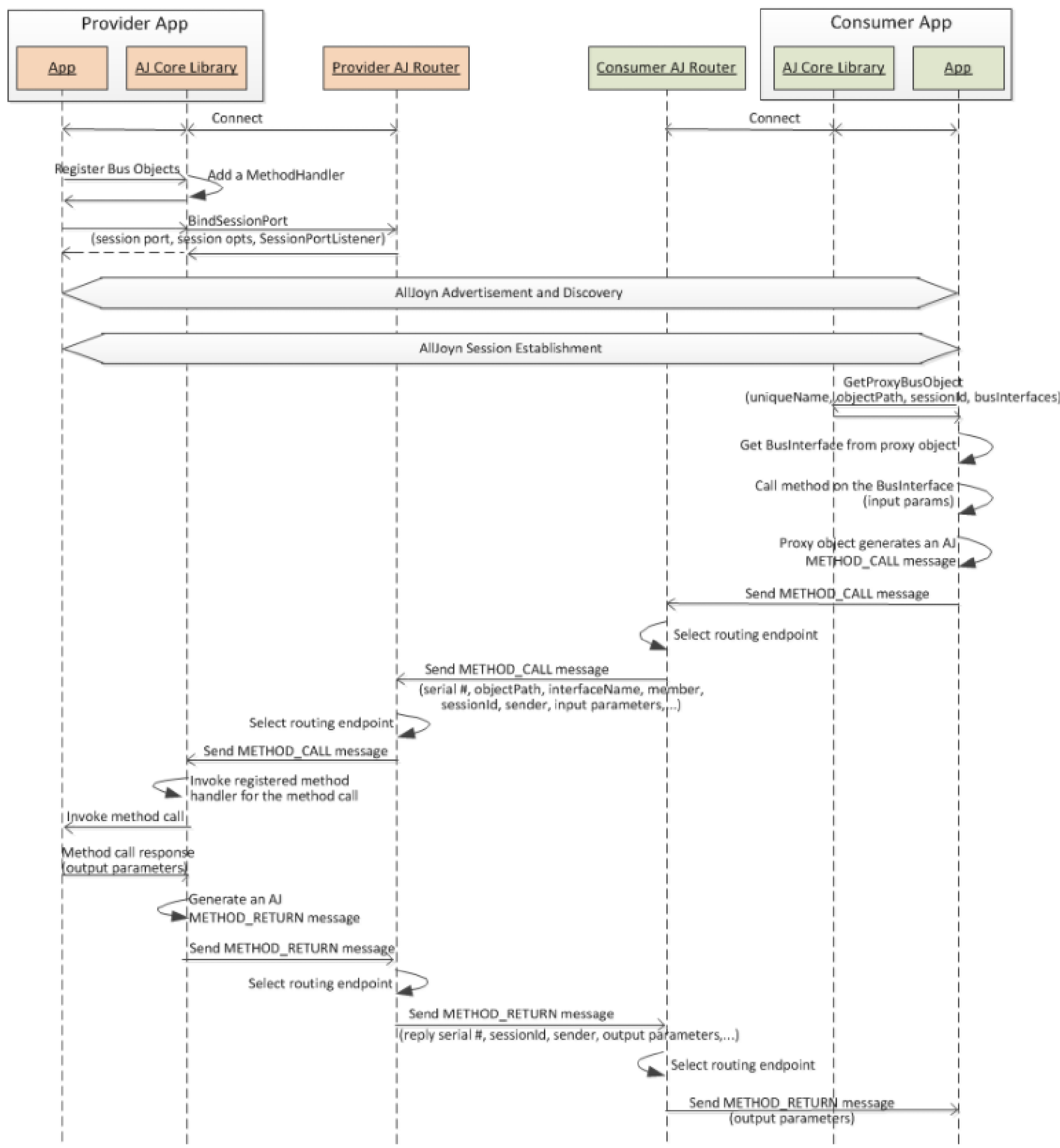

Figure 3.3: AllJoyn signal exchange for a Method call [8]

3.2.1 Methods

The EMS uses method calls for its control signals to ensure the signal is received by the desired DMS. The method reply does not currently contain any information about the device, but it would be possible to have the DMS reply with its participation level of the specified control signal. Table 3.3 describes each method with its name, units, and description of use.

Name	Units	Description
Import Power	Watts	set the power to be consumed from the grid by the DER for one hour
Export Power	Watts	set the power to be produced for the grid by the DER for one hour
Regulation	Watts	two second duration normalized power setting that is positive for import and negative for export

Table 3.3: DERAS methods interface to control DER

3.2.2 Properties

AllJoyn properties are variables that hold values, which can be *read*, *read-write*, or *write*. All properties are defined as *read only* since they represent physical values on a device that cannot be modified. For simplicity in data interpretation, all property values are set to the "double" data type. A double data type, requires 8 Bytes of memory, providing a maximum value of 1.7E $\pm$ 308. While this is inefficient for data transfer, it does allow the EMS a great deal of flexibility during testing. Table 3.4 describes each property with a name, units, and description. The chosen properties were designed in consideration to NERC recommendations. However, the properties were simplified to reduce the complexity of physical device implementation.

Name	Units	Description
Import Power	Watts	defines the max power that can be consumed from the grid
Export Power	Watts	defines the max power that can be produced for the grid
Import Energy	Watt-hours	defines the max power for a period of time that can be produced for the grid
Export Energy	Watt-hours	defines the max power for a period of time that can be produced for the grid
Import Char	Watts-per-second	defines the import power ramp rate
Export Char	Watts-per-second	defines the export power ramp rate
Idle Char	Watts-per-second	defines the available export energy loss rate

Table 3.4: DERAS property interface

3.2.3 Realized Distributed Management Systems for DERs

A DMS has been developed for two physical DERs, as well as a simulated DER for testing. The first DMS interface was designed for a BESS. Figure 3.4 shows the component representation of the class Unified Modeling Language (UML) for the BESS located within the Portland State University (PSU) Power Lab [20, 21, 22, 23, 24, 25].

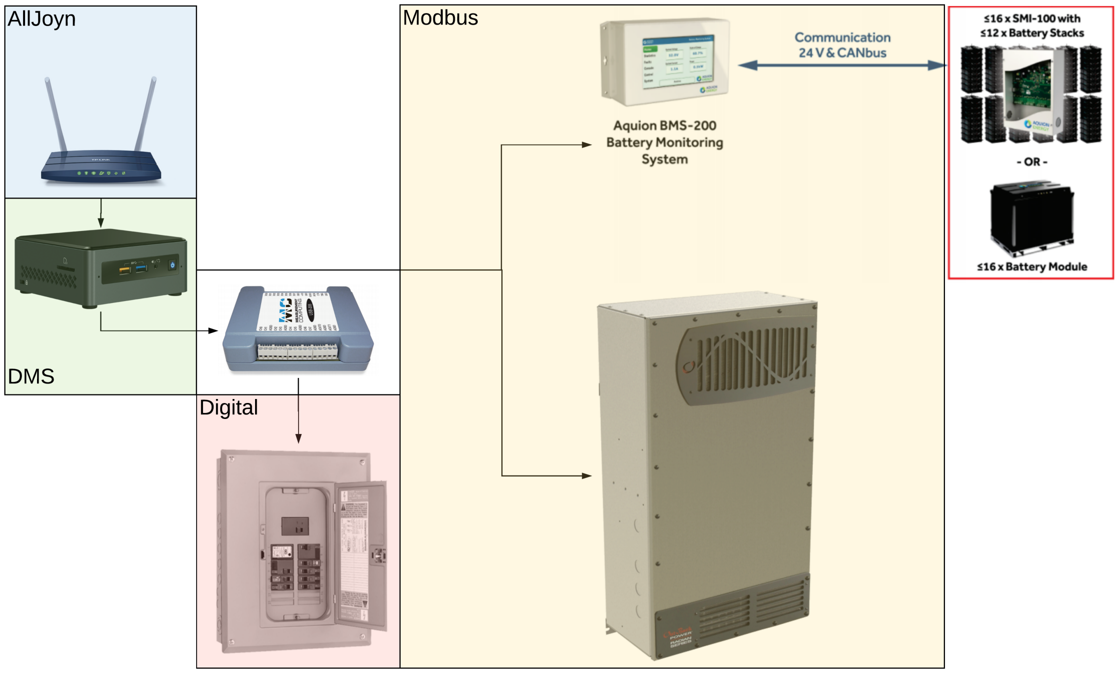

Figure 3.4: BESS component diagram

Figure 3.5 demonstrates the class UML. There are four interfaces for the BESS. The first interface is the AllJoyn signals from the EMS, which are interpreted by the AllJoyn DMS running on a small computer. The DMS translates the import/export power methods into the respective charge/discharge controls for the second and third interface. Each method calls Modbus and/or data acquisition (DAQ) methods to interface with their respective physical components. The inverter and battery monitoring system communicate via Modbus protocol. The smart load center and the H_2/Temperature sensor communicate via the DAQ with a high or low voltage signal on the digital input/output channels.

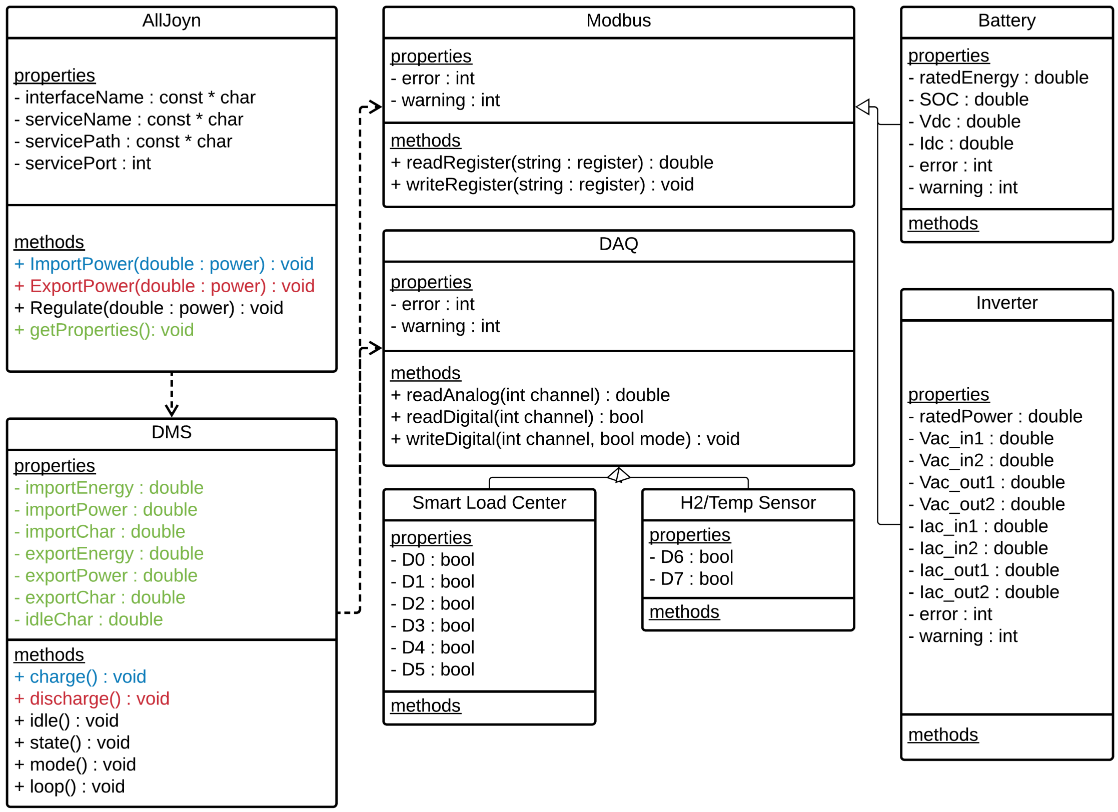

Figure 3.5: BESS class UML diagram of the interface between physical components

The second DER was a EWH, also located within the PSU Power Lab. The EWH component representation in Figure 3.6 shows the simple DMS interface for control [25, 26, 27].

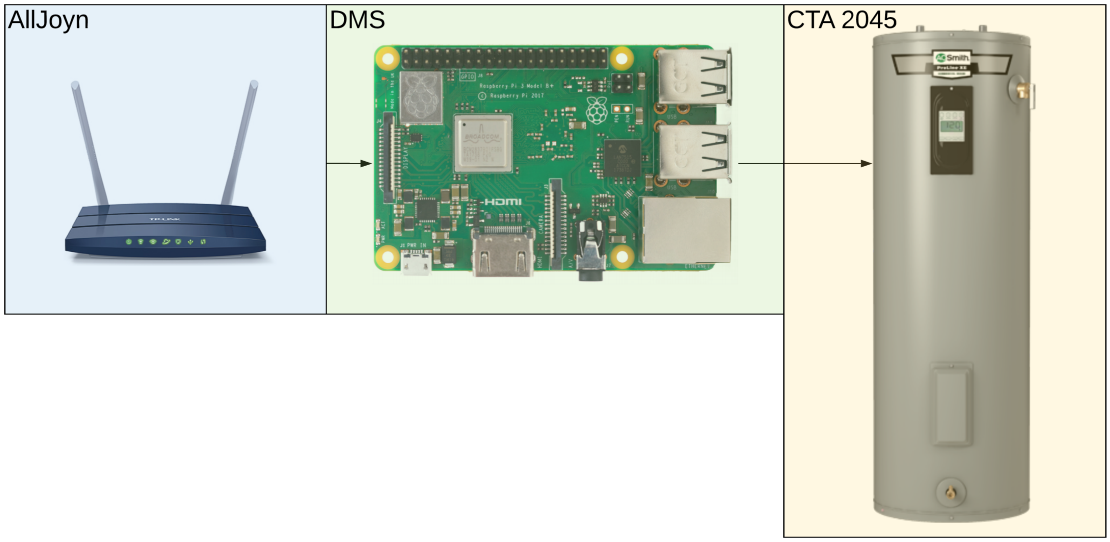

Figure 3.6: EWH component diagram

Figure 3.7 shows the class UML diagrams for the EWH, which has two interfaces. The EMS AllJoyn signals are interpreted by the AllJoyn DMS running on a Raspberry Pi, which translates the import/export power methods into the respective absorb/shed controls. The Raspberry Pi's general-purpose input/output (GPIO) are used to control power going to the water heater via CTA 2045 controls, allowing the EWH to shed or absorb.

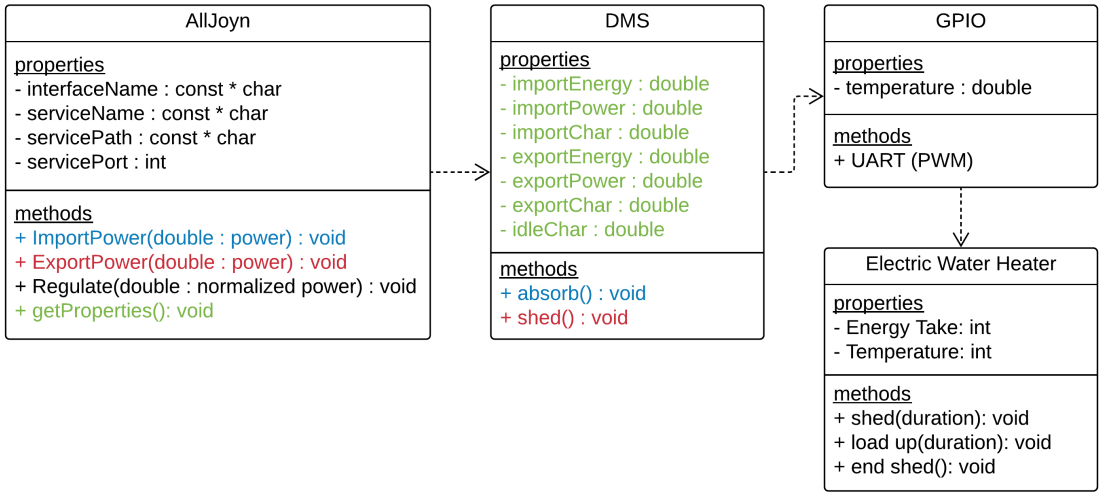

Figure 3.7: EWH UML diagram of the interface between physical components

The most challenging aspect of developing the DMS interface for DER is the device driver. Each DER has its own control standard with its own protocol specifications. VOLTTRON has developed drivers for BACNet, Modbus, SEP 2.0, and DNP3 devices. The DMS developed for this research support CTA 2045 for the electric water heater and Modbus for the BESS system. There are plans for implementing SEP 2.0 during the next revision of this research.

3.3 Network Architecture

The network communication uses a Virtual Private Network (VPN) to extend the Local Area Network (LAN) to create a Wide Area Network (WAN) for device-device communication. The LAN and WAN are communication links that allow devices to communicate with each other. The only differentiation is the geographical distance between the devices that dictates a reclassification of the network. Figure 3.8 demonstrates the VPN tunnel over the WAN.

DERAS was designed to be a series of network bridges using OpenVPN tunnels to create a central network.

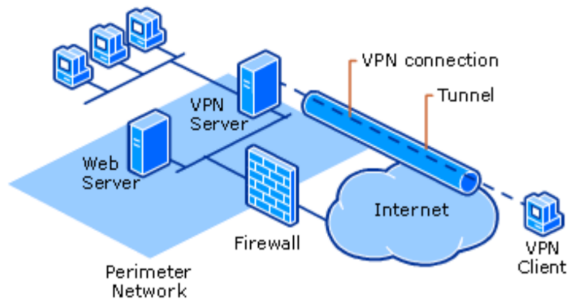

Figure 3.8: VPN tunnel between client/server networks [9]

A VPN doesn't physically create a tunnel between the two networks, but instead uses encryption to scramble the transferred data so that only the desired device may receive the packet and be able to understand it. We decided to use a VPN for the following reasons [9]:

- PGE and VOLTTRON referenced a VPN as a secure way to connect devices over the internet.
- A VPN provides defense against cyber threats throughout the network's devices regardless of location.
- A VPN provides access to internal network services by bridging the physical connection. (i.e., AllJoyn routing).
- A VPN can authorize network access by users reducing the risk of bad actors connecting to the aggregator.

Figure 3.9 shows AllJoyn's LAN communication bus. Each device (i.e., server, NUC, Raspberry Pi) can support multiple applications (i.e., DMS, EMS) running over the open-source D-Bus protocol. D-Bus was designed to allow applications to seamlessly communicate with each other over the device application layer. AllJoyn's router provides the bridge for device-device communication over the LAN.

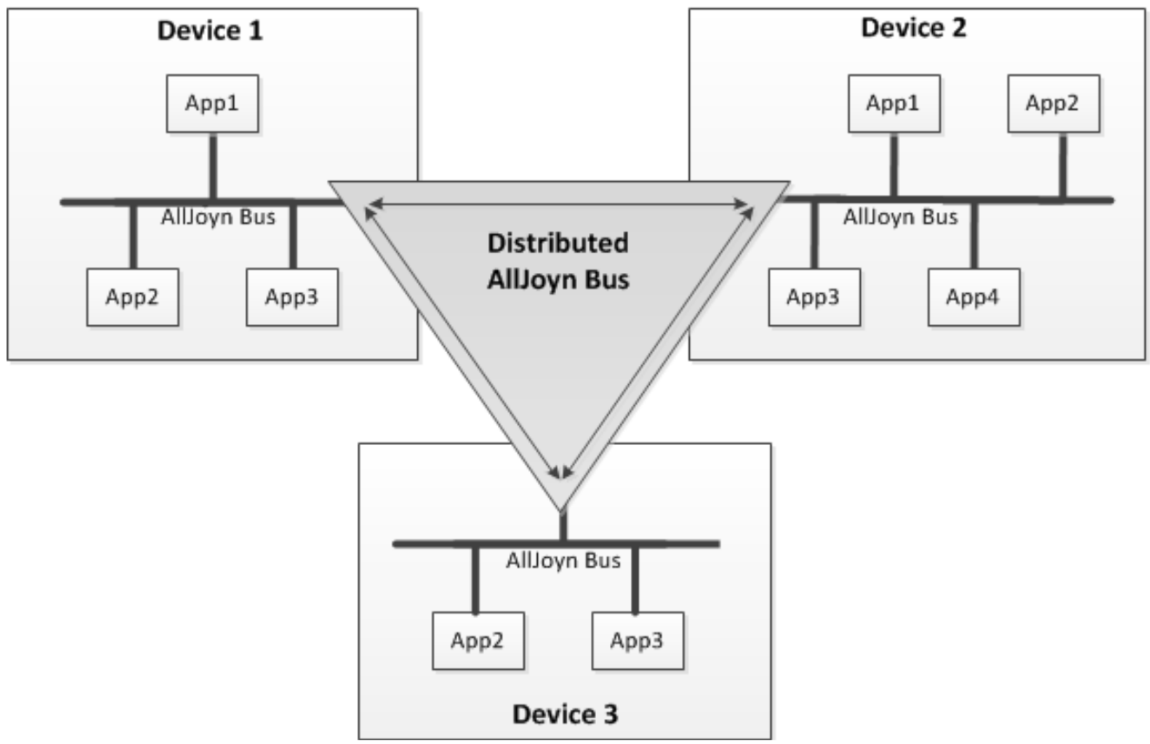

Figure 3.9: Distributed AllJoyn Bus [8]

3.4 Emulation

Appropriate modeling methods will be required to support mass integration of DERs into the BPS. The DER models are used to understand resource availability and to reduce the data transfer requirements to maintain visibility and control. This research uses simplified linear models of the connected DER to approximate the behavior of the aggregated DER. Each hour, the EMS collects the property values from every connected DMS to update the aggregated digital twins and update the next hour dispatch schedule. Using the digital twins, the EMS can receive dispatch schedules and update the models as dispatch changes over the hour without having to get properties from the physical DER every time dispatch changes.

NERC recommends the equivalent models for DER aggregation support non-uniform parameters between various DER and consider diversity of the PCC voltage of DER at various locations on the distribution feeder. The current models do account for non-uniform parameters such as the ramp rate for import/export power control, but they do not model the PCC voltage over the projected hour. If the PCC was adopted as a property for DER, it would be added to the reply message for method calls. This would allow the EMS to make better informed decisions when aggregating DERs for dispatch.

3.4.1 Battery Energy Storage System

Figures 3.10 and 3.11 show the volt-energy curves for charging and discharging of the Aquion battery used by our BESS system. The simulated DER will use the rate of charge/discharge and its present State of Charge (SoC) to determine the import/export energy available to the EMS. The import/export power available is determined by the lowest rating of the battery or inverter. The BESS system at PSU is currently limited to the electrical service ratings for its branch circuit.

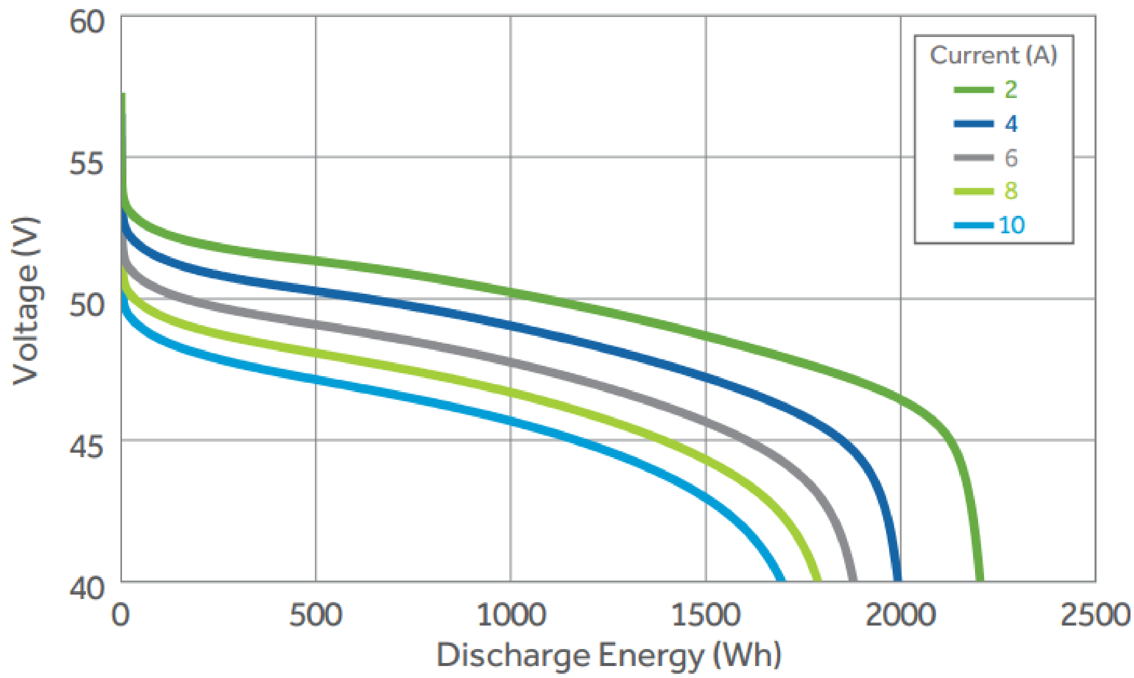

Figure 3.10: Aquion discharge profile for various rates over a ten hour period [10]

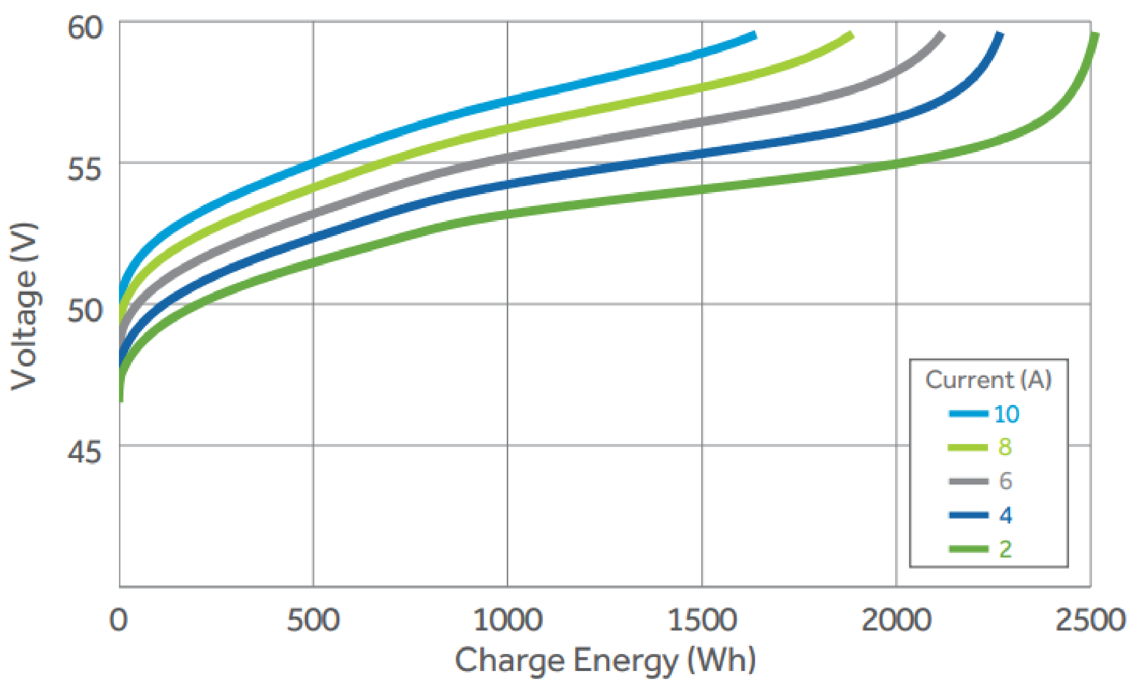

Figure 3.11: Aquion charge profile for various rates over a ten hour period [10]

Figure 3.12 shows the charge/discharge response from the BESS. This ramp response rate sets the import/export characteristics so the EMS may emulate their behavior between the hourly telemetry calls. The idle characteristic is determined by the idle SoC loss of approximately 1% a day, which converts to 0.1 Wh per second.

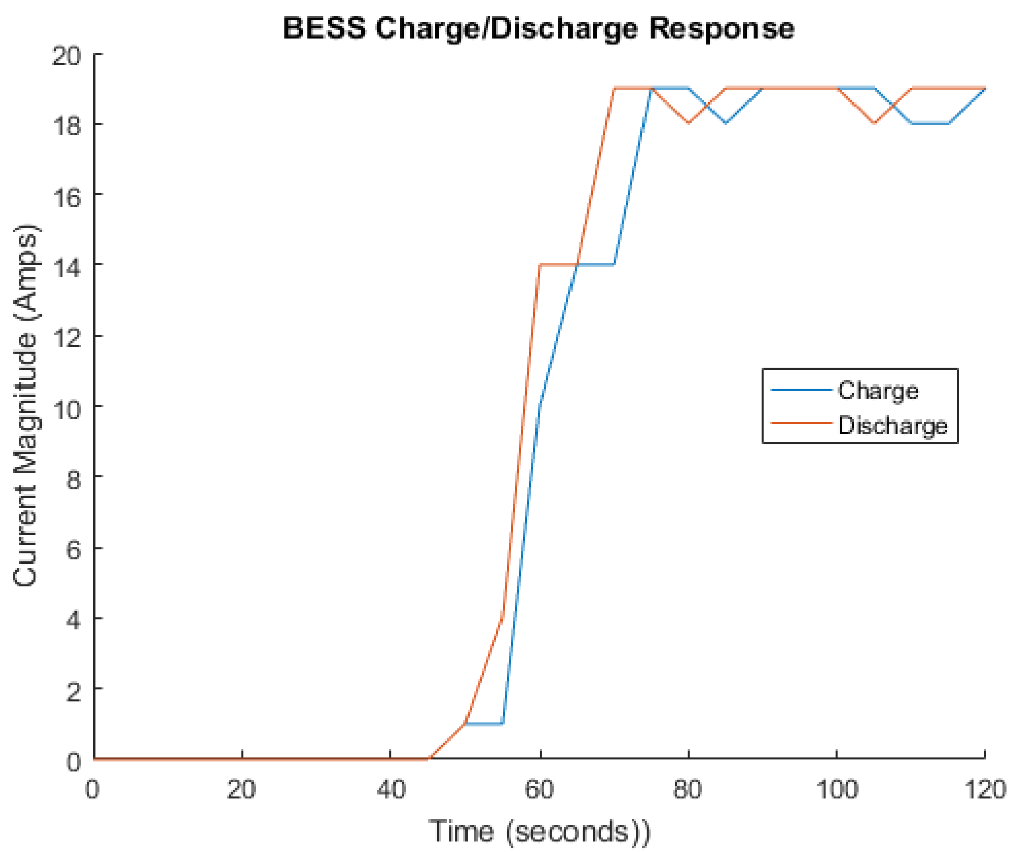

Figure 3.12: BESS charge/discharge response to control signal

Figure 3.13 demonstrates the reduced energy capacity available based on the charge/discharge power setting. The energy capacity is approximated using a linear relationship derived from the charge/discharge profiles seen in Figures 3.11 and 3.10. This relationship is used by the DMS to inform the EMS of the energy available between telemetry calls.

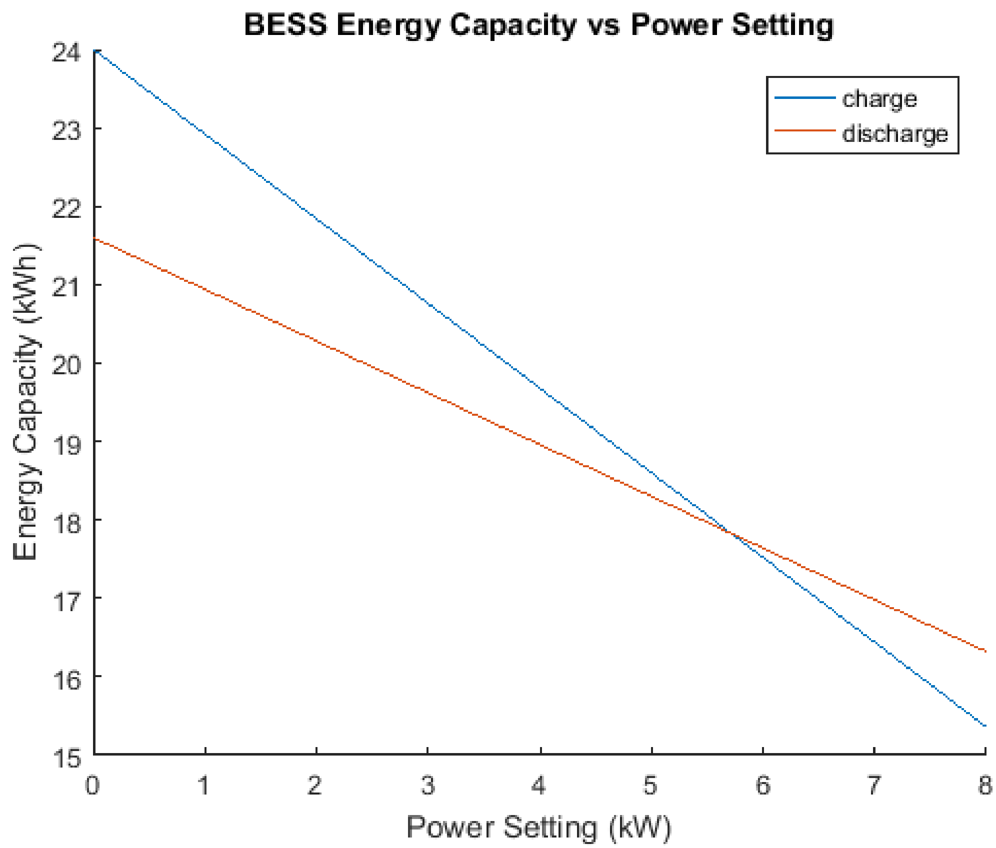

Figure 3.13: BESS energy capacity vs charge/discharge rate

Table 3.5 provides a summary of all the property values used to simulated the BESS at PSU. The import/export energy is updated in response to every regulation control signal from the EMS. The import/export energy is is not reported to the EMS until the hourly telemetry property call.

Property	Value	Units
Import Power	3000	Watts
Export Power	3000	Watts
Import Energy	24000	Watt-hours
Export Energy	21500	Watt-hours
Import Char	100	Watts-per-second
Export Char	100	Watts-per-second
Idle Char	0.1	Wh-per-second

Table 3.5: BESS simulated model properties summary

3.4.2 Electric Water Heater

Unlike the BESS, the EWH energy availability is determined by the temperature of the tank. Figure 3.14 shows the effects of water draw on the energy take, or energy needed to return the tank's temeprature to the consumer operating limits. When a consumer draws water from the tank, it is replaced with cool water from the inlet valve. The cool water takes time to mix with the warm water and be sensed by the internal temperature sensor.

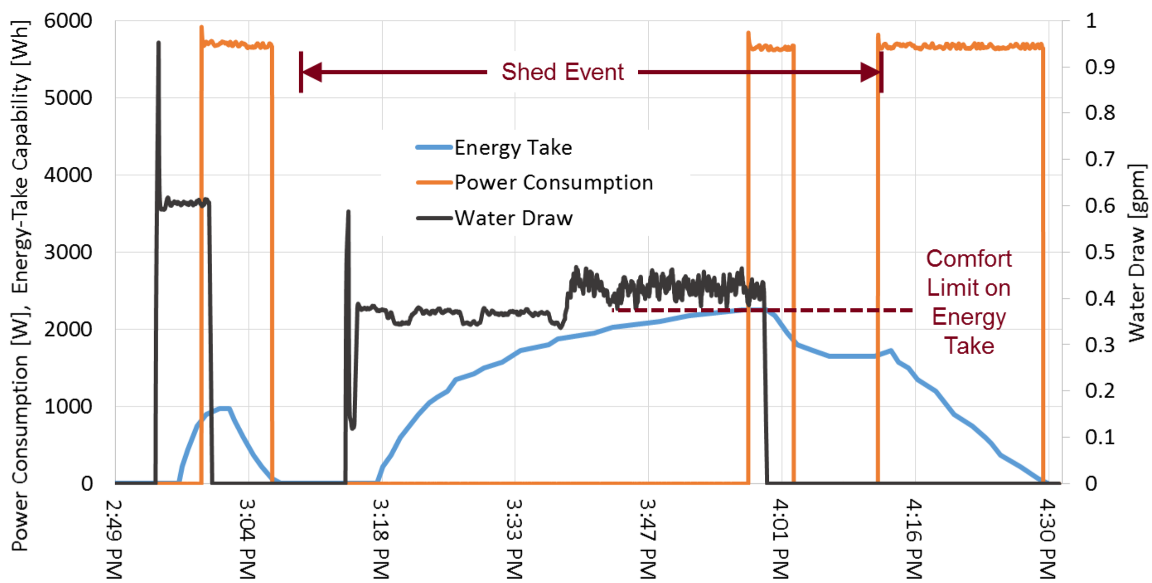

Figure 3.14: EWH energy capacity vs water draw and power consumption [11]

Figure 3.15 displays a water draw profile for an EWH, establishing the energy available to shed from the grid. Forecasting the daily usage profile for electric water heaters is outside the scope of this research. The water draw profile introduced is the only profile used by DERAS to estimate the available energy.

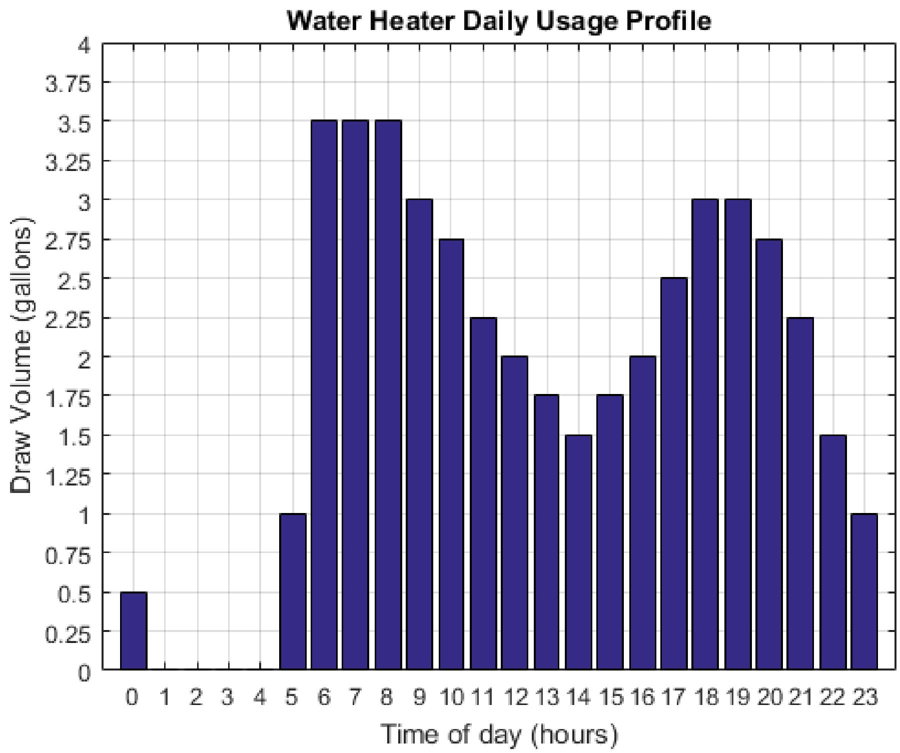

Figure 3.15: EWH daily user profile [12]

Annie Clark in [12], developed a model for an EWH based on the tank volume, internal temperature, and inlet temperature. Using the convention that absorb/shed is equivalent to import/export respectively, the EWH properties can be translated to the interface described by DERAS. The response time for EWH is considered to be instant due to the rapid transient response of resistive elements.

The idle characteristic losses are driven by the temperature differential between the EWH and the environmental air temperature. Clarke found the loss function to be approximately 0.528 (Watts-per-^oF) for the EWH located at PSU. Table 3.6 displays the properties used to simulate the EWH. The import/export energy for the EWH is determined by assuming the user profile for water draw within the next hour is accurate.

Property	**Value**	**Units**
Import Power	1500	Watts
Export Power	1500	Watts
Import Energy	1300	Watt-hours
Export Energy	1300	Watt-hours
Import Char	1500	Watts-per-second
Export Char	1500	Watts-per-second
Idle Char	1	Watts-per-second

Table 3.6: EWH simulated model properties summary

3.5 PJM's Regulating Control Signals

PJM Interconnection is an RTO that coordinates the movement of wholesale electricity in all or parts of Delaware, Illinois, Indiana, Kentucky, Maryland, Michigan, New Jersey, North Carolina, Ohio, Pennsylvania, Tennessee, Virginia, West Virginia and the District of Columbia [28]. PJM uses two control signals to balance the constantly changing loads and generation within its operating region.

The control signals RegA and RegD are designed to be fast acting control signals for frequency regulation. The bulk power system within PJM's operating region operates at a frequency of 60 Hertz. If there is too much generation feeding the BPS, the frequency will begin to increase. If the load becomes greater than the generation feeding the BPS, the frequency will begin to decrease.

Figure 3.16 shows the RegA control signal, defined as a low filter signal for traditional regulating resources [13]. The ramping rates required to follow a RegA control signal are much lower than for RegD.

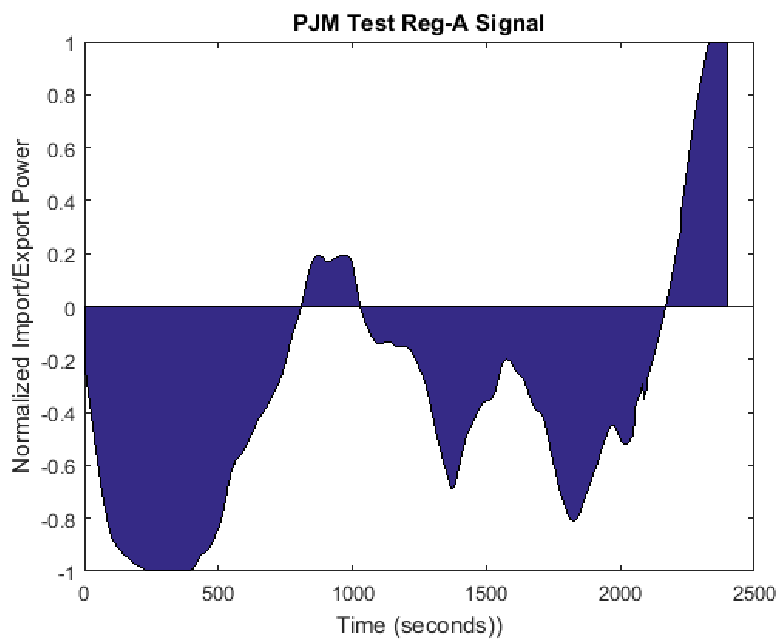

Figure 3.16: PJM's RegA test control signal [13]

Figure 3.17 shows the RegD control signal, defined as a high filter signal for dynamic regulating resources. The ramp rates are significantly greater than RegA, with more frequency changes between importing and exporting power. RegD is a symmetric signal, ensuring a regulating resource will import the same energy that it exports over each hour. Both RegA and RegD control signals are sent to participating regulating resources every two seconds.

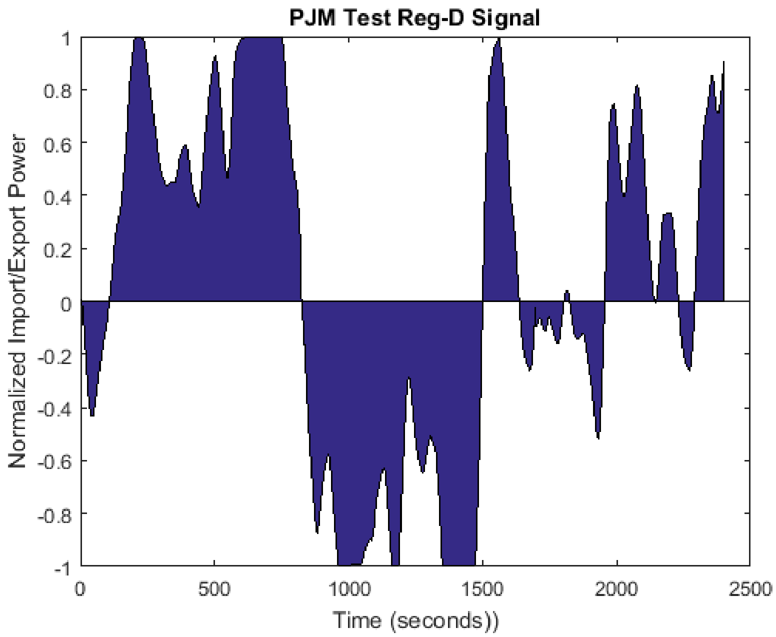

Figure 3.17: PJM's RegD test control signal [13]

PJM's regulation market obligations state that resources assigned must be capable of responding to either RegA or RegD control signals immediately and meet their capability within five minutes [13]. Resources participating in PJM's regulation market must also maintain the required performance characteristics, which will be discussed in the following sections.

3.5.1 Resource Owner Requirements

PJM requires the total regulation and the current regulation information for all participating resources. PJM has designated these signals as follows:

- **Total Regulation (TRegA or TRegD):** The resource regulation capability (MegaWatts) that represents the resource's ability to regulate. This signal must be sent every two seconds to PJM.

- **Current Regulation (CRegA or CRegD):** The resource feedback ($\pm$ Mega-Watts) representing the active position of the resource with respect to the total regulation signal. This signal must also be sent every two seconds.

3.5.2 Performance Qualifications

A resource must pass three performance compliance tests administered by PJM to be eligible to participate in the regulation market. After a generation resource has been approved as eligible with a performance of at least 75%, it must maintain a performance level greater than 40%, else it will have to re-apply to participate in the regulatory market. The performance average is reset after 100 consecutive hours of participation. Equations 3.2-3.5 in the following sections express PJM's performance criteria.

3.5.2.1 Delay and Correlation Score

Equation 3.1 calculates the delay score for each ten second interval for each δ between 0 to 10 seconds. Equation 3.2 uses a linear regression function (r) to quantify the relationship between the resources response and the control signal. The maximum sum of the delay and the correlation results in the δ time offset that will be used for the actual performance calculation. During times where the standard deviation of the regulation signal is less than the threshold value, the correlation will be determined using Equation 3.3.

$$Delay = |\frac{\delta - 5_{minutes}}{5_{minutes}}| \tag{3.1}$$

$$Correlation = r_{signal}(\delta, \delta + 5_{minutes}) \tag{3.2}$$

$$Correlation = 1 - |\Delta_{slope}| \tag{3.3}$$

3.5.2.2 Precision Score

Equation 3.4 is used to compare the energy provided by the resource to the energy requested. The average response over a ten second period is then averaged on an hourly basis to determine the response error.

$$\begin{aligned} &PrecisionScore = 1 - \frac{1}{n}\sum |Error| \\ &where, \\ &Error = |\frac{response - signal}{signal_{hourly}}| \end{aligned} \tag{3.4}$$

3.5.2.3 Performance Score

Equation 3.5 shows the final calculation to determine the performance of a resource. The score is averaged over a five minute period, where any periods without assigned regulation are removed. The performance is then averaged over an hour period. The Delay, Correlation, and Precision are all weighted equally for the performance calculation.

$$
\begin{aligned}
Performance(t) &= max_{i=0->5}([X+Y]+Z) \\
where, \\
X &= A \cdot Delay(t+i) \\
Y &= B \cdot Correlation(t+i) \\
Z &= C \cdot Precision(t) \\
A, B, C &= \frac{1}{3}
\end{aligned}
\tag{3.5}
$$

4 Results & Analysis

The following section presents DERAS's network performance for increasing DMS connections. Ten DMS were simulated before starting the twenty second sample RegD control signal. This cycle was repeated until the total number of simulated DMS reach 250. The performance was measured by: physical server CPU and memory usage, overall data transfer, and network traffic delay times. Additionally, the simulated DER model response to the sample RegD signal and PJMs performance equation were used to assess the regulating capability of aggregated DERs.

4.1 Network Testing

The network tests were designed to gain insight into the scalability of DERAS and validate the use of the AllJoyn IoT framework for visibility and control of DERs. The test flowchart is shown in Figure 4.1. A twenty second sample RegD control signal, ten total values, was used to test DERAS's ability to maintain visibility and control of each DER while maintaining the two second signal rate of the control signal. The server performance and network traffic was recorded for thirty seconds, starting just before the test was executed.

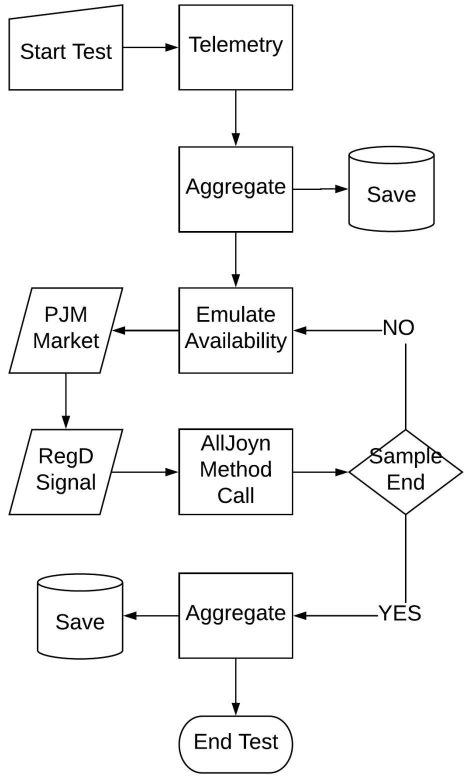

Figure 4.1: Flowchart for DERAS sample RegD control test

Figure 4.2 shows the testing procedure used. The test setup consisted of five Raspberry Pi's located on a single LAN. Each test consisted of ten simulated DMS connected to the EMS via one of the Raspberry Pis, through an OpenVPN tunnel. After ten new Distributed Management Systems (DMSs) were connected, the EMS would start the sample RegD control test and record the performance. Once the test was complete, ten more DMSs would be simulated on the next Raspberry Pi. This cycle was repeated until each Raspberry Pi was simulating 50 DMSs.

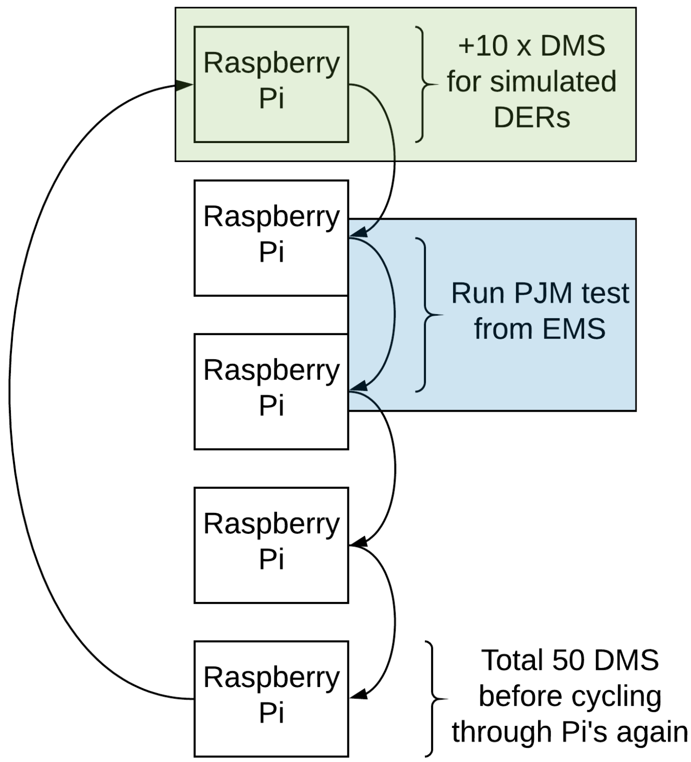

Figure 4.2: Testing procedure for simulated DMS

The simulated DMSs are located at PSU and the EMS server is located in Beaverton, Oregon, approximately twelve miles apart. The difference in physical location of the systems ensures realistic network signal delay with DERs located at various residential homes.

Figure 4.3 shows the network traffic of DERAS at 10, 100, and 250 DER connections. The larger 500 byte spikes at the beginning and end of the ten connection test highlight the telemetry data from each DMS, while the smaller 300 byte spikes are PJM's regulation control signals. Further inspection of the plot shows the regulation control signal spikes are evenly space at approximately two second intervals.

The 100 connection test shows a ten-fold increase in the signal delay to process each property call and regulation control signal. The delay causes the end telemetry data to fall outside the thirty second network traffic collection period. The regulation control signal period still appears to be twenty seconds in duration, so there was no extensive delay between control calls.

The 250 connection test demonstrates a huge delay and the ineffectiveness on DERAS's ability to maintain visibility and control of aggregated DERs. The EMS had to wait on 250 requests for telemetry, followed by 250 responses with the telemetry, which was approximately 18 seconds. The delay was so long that the end telemetry call wasn't recorded within the thirty second test. Dividing 500 total signals by the total time to send and receive yields a signal delay of nearly 28 milliseconds. This results in 6 seconds round trip to send each RegD control signal to each simulated DER, which would dramatically affect the PJM's performance criteria.

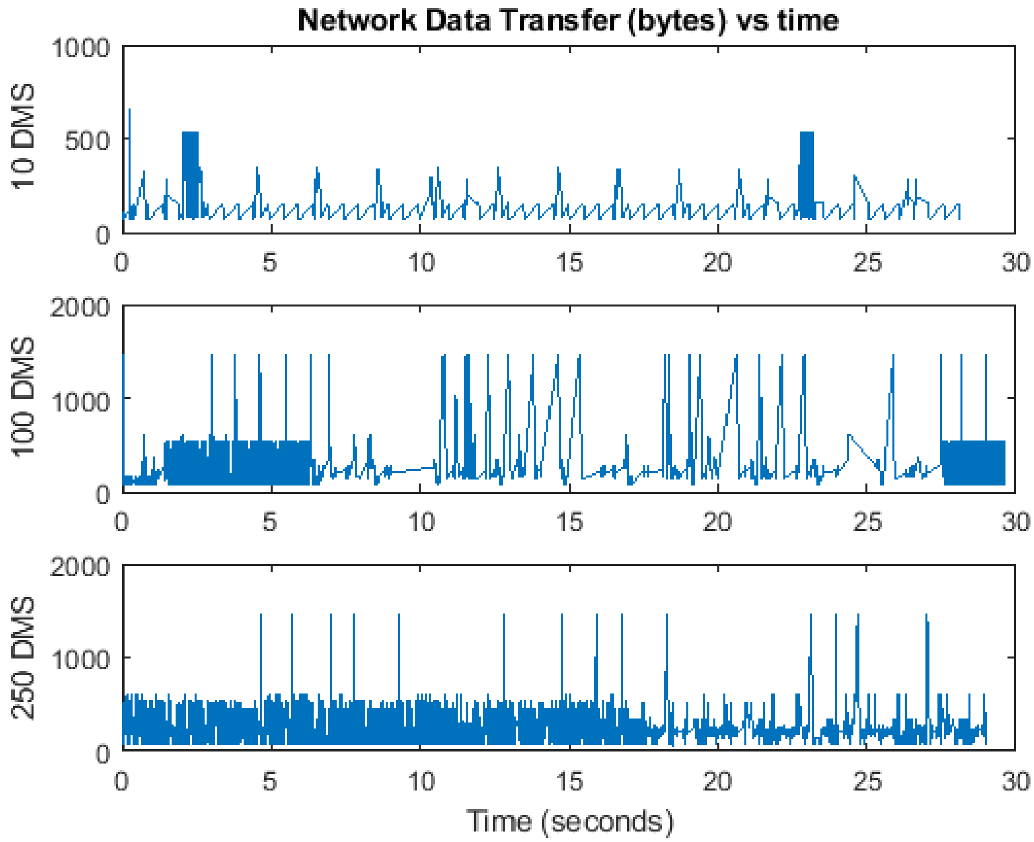

Figure 4.3: DERAS network data signals for 10, 100, and 250 simulated DMS

Figure 4.4 shows how the network traffic scales with incremental increases in connections. The expected data transfer rate was a small linear relationship with each increase in connected DMS. The data transfer rate had an exponential increase for each connected DMS. It appears to become linear after the first 100 connections, but this is because the test procedure did not capture the end telemetry calls.

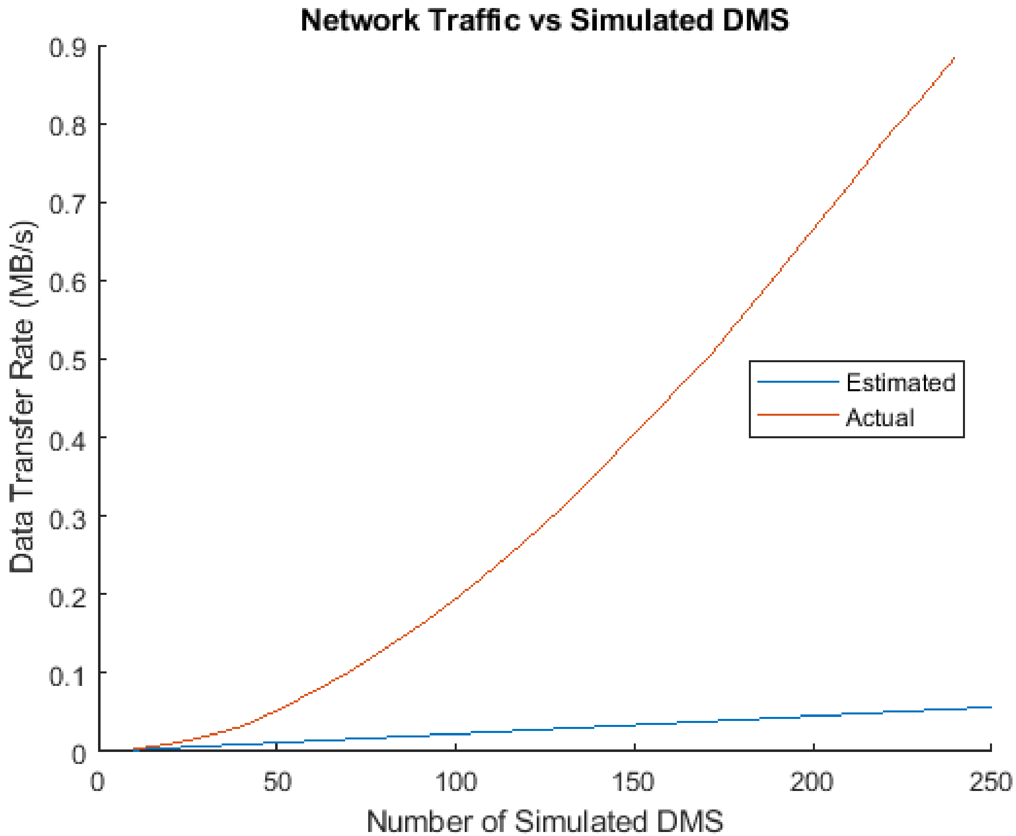

Figure 4.4: DERAS total data transfer for 0-250 simulated DMSs

The performance of the server is shown in Figure 4.5. The EMS design constraints were difficult to define for a network server attempting to aggregate a large number of connected devices using an unoptimized prototype application and a VPN connection. The 250 connection tests uses less than 2.5% of the allocated memory and less than 1% of the allocated CPU. The memory requirements increase linearly for each connected DMS in order to process the telemetry data and emulate the aggregated models. The CPU usage appears to be volatile, which is likely caused by inefficient application process calls of the EMS core code, but its usage is almost non-existent for the limited device connections.

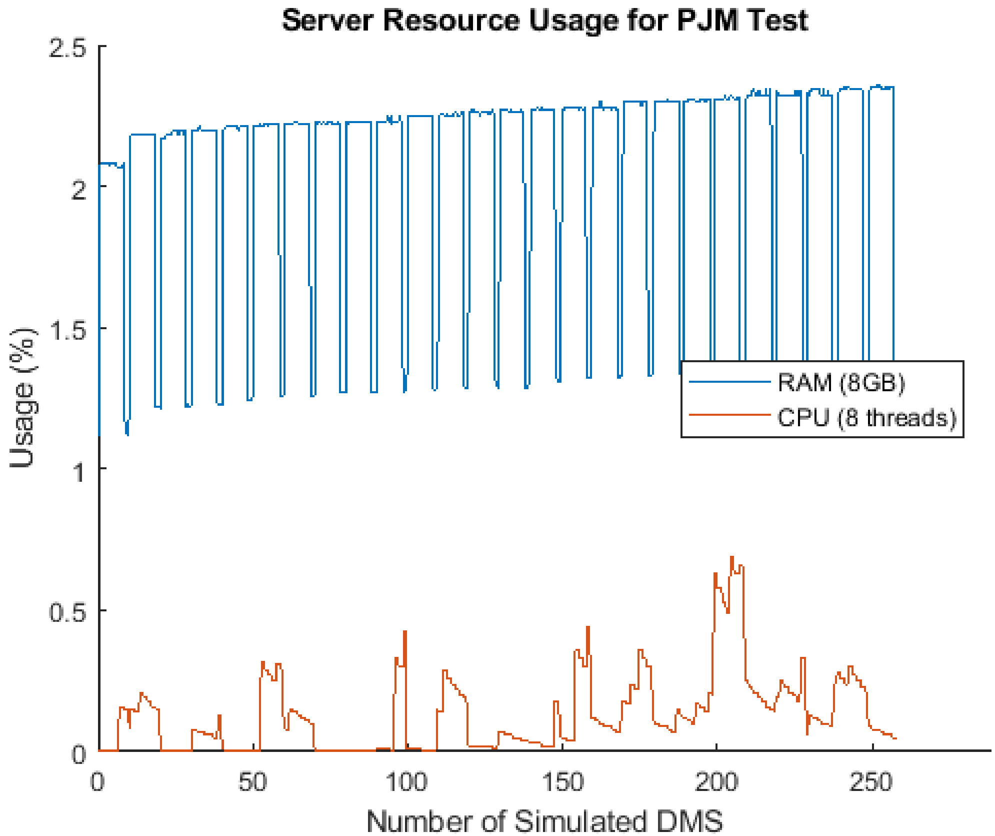

Figure 4.5: DERAS client CPU/RAM usage for 0-250 simulated DMS

4.2 PJM Regulation Performance

Figure 4.6 shows the aggregated performance of ten Battery Energy Storage Systems (BESSs) at approximately 97% for the RegA test signal. The BESSs lose approximately 10% of their available energy over the test period because the RegA control signal is not symmetric. If the control signal was symmetric over the test period, the BESSs would have imported the same energy as they exported.

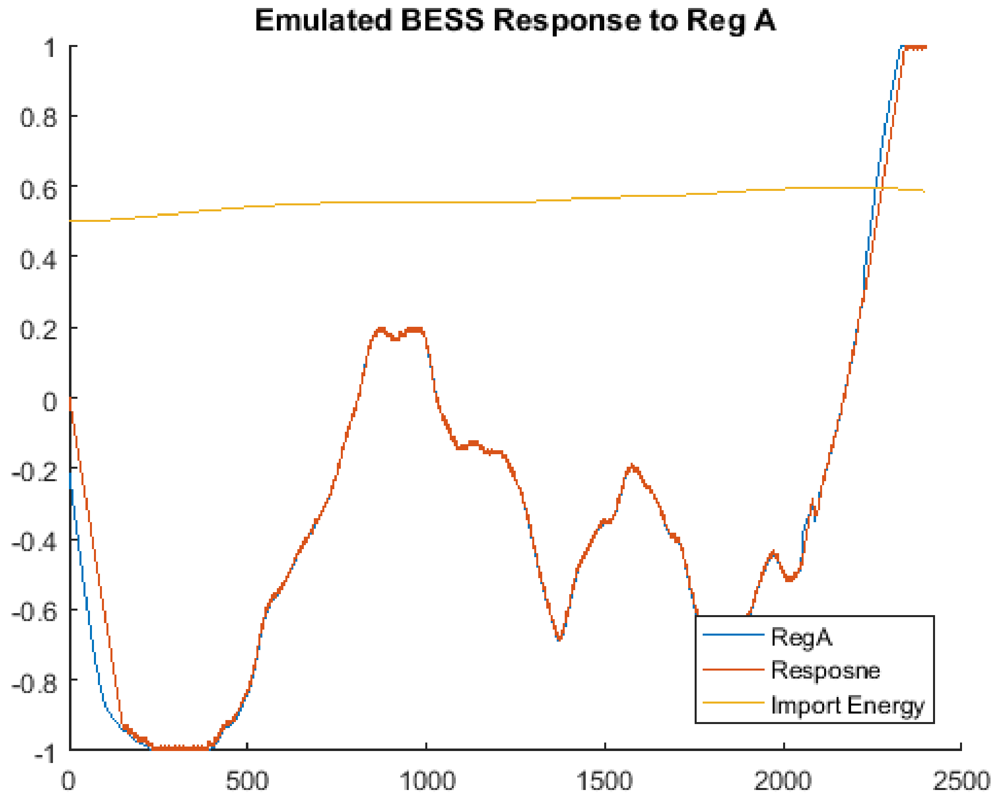

Figure 4.6: PJM RegA control signal performance for aggregated BESS

Figure 4.7 demonstrates the aggregated performance of ten BESSs at approximately 88% for the RegD test signal. The performance of the BESS was much higher than anticipated considering the slow ramp characteristics of the model used. The ten BESSs all shared the same characteristics for this performance test, but the performance of the aggregated system will improve as we add BESSs with faster ramp capabilities.

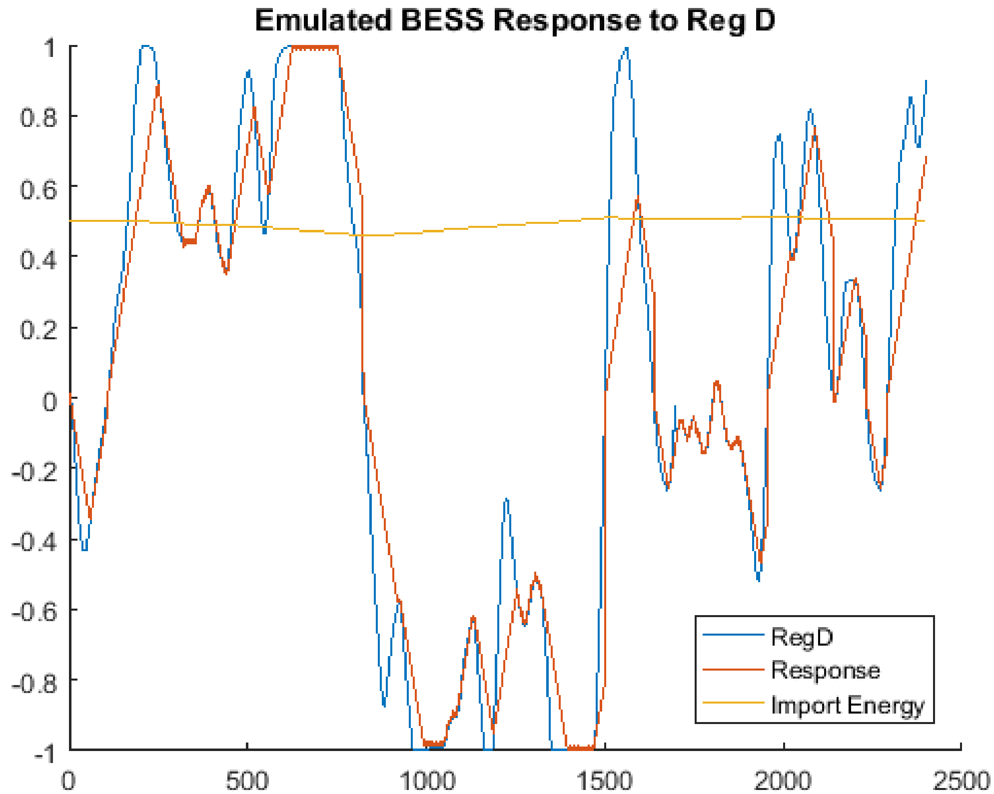

Figure 4.7: PJM RegD control signal performance for aggregated BESS

Figure 4.8 shows the aggregated performance of ten EWHs for the RegD test signal. PJM's performance qualification resulted in a score of 97%, well above the 75% performance criteria. EMS assigned a percentage of available EWHs with the greatest available energy to achieve import power control. This dispatch method is most beneficial when the number of aggregated resources is very large. The emulated performance is highly dependent on the forecasts of the residential water draw. If the forecast is incorrect, then the water heater will not be able to participate in the dispatch signal.

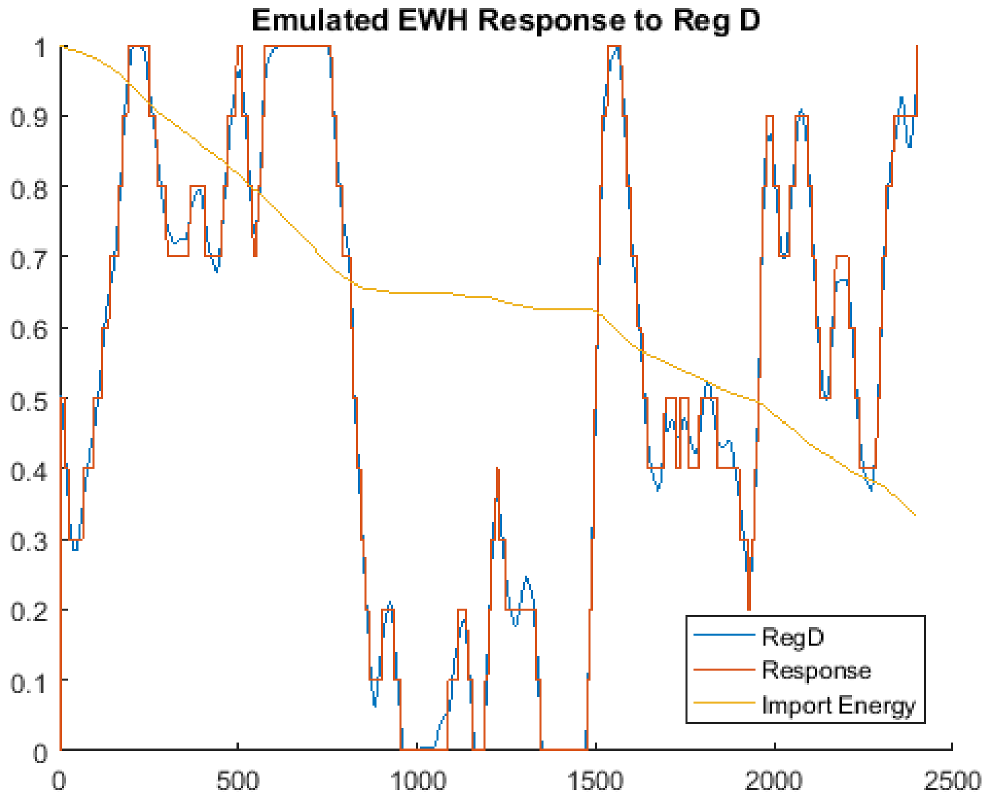

Figure 4.8: PJM RegD control signal performance for aggregated EWH

5 Discussion

The testing performed on DERAS exposed three major issues with the current implementation. The first issue is the AllJoyn connection limit, which prevents the simultaneous connection of more than a few hundred applications. This did not appear to be an issue during the initial investigation of the connection limitation of the AllJoyn router. The stand-alone AllJoyn router XML, found in Appendix A.1, allows the number of connections to be set to thousands and there was no direct discussion of session limitations within the AllJoyn documentation.

The second issue is how the EMS gathers telemetry data and calls import/export control methods. All telemetry and method calls were designed to be sequential. The compounding delay would not have made an impact on hourly dispatch schedules, but PJM's RegA and RegD control signal's two second rate makes it vital to process all dispatch signals before the next signal is received.

There are two possible solutions to this issue. One is to reduce visibility and use AllJoyn's signals to broadcast control to connected devices. The EMS could send a single signal to control all devices with this technique, but many optimization options would be lost because each asset could no longer be controlled individually.

The other possible solution is to parallelizing the AllJoyn method calls. The EMS could spawn an application thread to send an import, export, or telemetry signal for a single DER.

This could be repeated for all connected DER, resulting in single observed signal delay. However, the number of packets per second, CPU, and memory usage would all increase.

The final issue is the adoption of AllJoyn as the framework to connect DERs. AllJoyn was acquired by OCF, which has ceased development on AllJoyn. The AllJoyn support wiki is no longer available and there is no activity on its forums. Microsoft has dropped its endorsement of AllJoyn in favor of OpenThread and the few consumer products that supported AllJoyn no longer do. It may be time to revisit other frameworks to maintain support and the benefits of consumer product development for DERAS.

6 Conclusion

DERAS was able to connect to remote DERs for energy regulation through PJM's RegA and RegD dispatch control signal. However, DERAS was limited to 250 connected DMSs. The aggregated DER's emulated performance of 10 agents was well within PJM's performance requirements for regulating resources.

The future work on DERAS will include machine learning techniques for DER classification, emulation, and optimization for dispatch. Additionally, use of aggregated models in steady-state and transient power flow studies will help to understand the effects of DERs on BPS reliability and resiliency. Emulation validation of individual DERs and PJM performance validation techniques will need to be defined.

The most important work for DERAS will be increasing the scale of aggregated devices and the decision of which IoT framework will provide the best support and scaling for DERAS. This will require an improvement to the WAN communication method. Using a VPN connection was a simple solution to extend AllJoyn's functionality, but a thorough review of current methods of WAN communication techniques will need to be carried out before large scale aggregation becomes feasible.

Bibliography

[1] NERC, "Distributed Energy Resources Connection Modeling and Reliability Considerations," tech. rep., February 2017. Available at http://www.nerc.com.

[2] International Energy Agency, "Renewables 2017 Analysis and Forcasts to 2022," tech. rep., October 2017. Available at https://www.iea.org/publications/renewables2017/.

[3] Galen L. Barbose, "U.S. Renewables Portfolio Standards: 2017 Annual Status Report," tech. rep., July 2017. Available at https://emp.lbl.gov/publications/us-renewables-portfolio-standards-0.

[4] GE Energy, "Western Wind and Solar Integration Study," tech. rep., May 2010. Available at https://www.nrel.gov/docs/fy10osti/47434.pdf.

[5] NERC, "2016 Probabilistic Assessment," tech. rep., March 2016. Available at http://www.nerc.com.

[6] S. Katipamula and J. Haack and G. Hernandez and B. Akyol and J. Hagerman, "VOLTTRON: An Open-Source Software Platform of the Future," *IEEE Electrification Magazine*, vol. 4, pp. 15–22, Dec 2016.

[7] "IoTivity Architecture Overview," April 2018. Available at https://iotivity.org/documentation/architecture-overview.

[8] "AllJoyn System Description," April 2018. Available at https://identity.allseenalliance.org.

[9] OpenVPN, "Securing Remote Access Using VPN," tech. rep., February 2018. Available at https://openvpn.net/WhitePaper.pdf.

[10] "Aquion Aspen 48S-2.2 Battery Specification Sheet," May 2018. Available at https://info.aquionenergy.com/hubfs/01_Product_Documentation/Aquion_Energy_Aspen_48S-2.2_Product_Specification_Sheet.pdf.

[11] EPRI, "Performance Test Results: CTA-2045 Water Heater," tech. rep., November 2017. Testing Conducted at the National Renewable Energy Laboratory. EPRI, Palo Alto, CA: 2017. 3002011760.

[12] Annie Clarke, "Electric Water Heater Modeling for CTA-2045 Demand Response Testing," Master's thesis, Portland State University, May 2018. Thesis.

[13] PJM, "PJM Manual 12: Balancing Operations," tech. rep., November 2017. Available at http://www.pjm.com/~/media/documents/manuals/m12.ashx.

[14] Hawaii State Energy Office, "Hawaii Energy Facts & Figures," tech. rep., May 2017. Available at https://energy.hawaii.gov/wp-content/uploads/2011/10/HSEOFactsFigures_May2017_2.pdf.

[15] U.S. Department of Energy, "2015 Progress Report for OE ARRA Smart Grid Demonstration Program Aggregation of RDSI, SGDP, and SGIG Results," tech. rep., May 2015. Available at https://www.energy.gov.

[16] M. Osborn, "Experiences in aggregating distributed generation for system benefit," in *IEEE Power Engineering Society General Meeting, 2004.*, vol. , pp. 886–889 Vol.1, June 2004.

[17] "OpenThread Primer," April 2018. Available at https://openthread.io/guides/thread_primer/.

[18] "Windows IoT Core documentation," April 2018. Available at https://docs.microsoft.com/en-us/windows/iot-core/archive/alljoyn.

[19] NERC, "Integration of Variable Generation Task Force Summary and Recommendations of 12 Tasks," tech. rep., June 2015. Available at http://www.nerc.com.

[20] "Aquion BMS-200 Battery Monitoring System Installation & Operation Manual," May 2018. Available at https://www.altestore.com/static/datafiles/Others/Aquion%20BMS-200%20Install%20ManualB.pdf.

[21] "Outback Radian Series Inverter/Charger 8048a Installation Manual," May 2018. Available at https://ressupply.com/documents/outback/Radian_GS4048A-8048A_Installation.pdf.

[22] "MC USB-200 Series 12-Bit Multifunction DAQ Device," May 2018. Available at https://www.mccdaq.com/PDFs/specs/USB-200-Series-data.pdf.

[23] "Intel NUC Kit NUC7CJYH," May 2018. Available at https://www.intel.com/content/www/us/en/products/boards-kits/nuc/kits/nuc7cjyh.html.

[24] "QO and Homeline Load Centers and Enclosures," May 2018. Available at http://download.schneider-electric.com/files?p_Reference=1100CT0501&p_EnDocType=Catalog&p_File_Id=9609596644&p_File_Name=1100CT0501.pdf.

[25] "AC1200 Wireless Dual Band Router Archer C50," May 2018. Available at https://www.tp-link.com/us/products/details/cat-9_Archer-C50.html#specifications.

[26] "Raspberry Pi 3 Model B+," May 2018. Available at https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf.

[27] "ProLine® XE Electronic Display 50-Gallon Electric Water Heater," May 2018. Available at https://www.hotwater.com/Water-Heaters/Residential/Electric/ProLine/XE/Electronic-Display/ProLine-XE-Electronic-Display-PXNS-50/.

[28] "About PJM," May 2018. Available at http://www.pjm.com/about-pjm/who-we-are.aspx.

Appendix A: Stuff

A.1 AllJoyn Router XML

```
<busconfig>
    <type>alljoyn</type>
    <listen>unix:abstract=alljoyn</listen>
    <listen>tcp:iface=*,port=9955</listen>
    <listen>udp:iface=*,port=9955</listen>
    <limit name="auth_timeout">20000000</limit>
    <limit name="max_incomplete_connections">1000</limit>
    <limit name="max_completed_connections">10000</limit>
    <limit name="max_remote_clients_tcp">10000</limit>
    <limit name="max_remote_clients_udp">10000</limit>
</busconfig>
```

A.2 Energy Management System

A.2.1 main.cpp

```
/******************************************************************************
 *    Copyright (c) Open Connectivity Foundation (OCF), AllJoyn Open Source
 *    Project (AJOSP) Contributors and others.
 *
 *    SPDX-License-Identifier: Apache-2.0
 *
 *    All rights reserved. This program and the accompanying materials are
 *    made available under the terms of the Apache License, Version 2.0
 *    which accompanies this distribution, and is available at
 *    http://www.apache.org/licenses/LICENSE-2.0
 *
 *    Copyright (c) Open Connectivity Foundation and Contributors to AllSeen
 *    Alliance. All rights reserved.
 *
 *    Permission to use, copy, modify, and/or distribute this software for
 *    any purpose with or without fee is hereby granted, provided that the
 *    above copyright notice and this permission notice appear in all
 *    copies.
 *
 *    THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL
 *    WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED
 *    WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE
 *    AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL
 *    DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR
 *    PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER
 *    TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
 *    PERFORMANCE OF THIS SOFTWARE.
******************************************************************************/

#include <iostream>
#include <vector>
#include <cstdlib>
#include <string>
#include <sstream>
#include <ctime>
#include <algorithm>
#include <map>
#include <chrono>
#include <thread>

#include <alljoyn/Status.h>
#include <alljoyn/BusAttachment.h>
#include <alljoyn/Observer.h>
#include <alljoyn/Init.h>

#include "utility.h"

#define INTF_NAME "edu.pdx.powerlab.demo"
```

```
using namespace std;
using namespace ajn;
using namespace qcc;

Utility tools;

/* convenience class that hides all the marshalling boilerplate from sight */
class AssetProxy {
    ProxyBusObject proxy;
    BusAttachment& bus;
  private:
    /* Private assigment operator - does nothing */
    AssetProxy operator=(const AssetProxy&);
  public:
    map<string,double> properties;

    AssetProxy(ProxyBusObject proxy, BusAttachment& bus) : proxy(proxy), bus(bus){
        proxy.EnablePropertyCaching();
    }

    bool IsValid() {
        return proxy.IsValid();
    }

    QStatus ImportPower(double t_watts) {
        cout << "Sending desired export of: " << t_watts << endl;
        Message reply(bus);
        MsgArg arg("d", t_watts);
        QStatus status = proxy.MethodCall(INTF_NAME, "ImportPower", &arg, 1, 0);
        return status;
    }

    QStatus ExportPower(double t_watts) {
        cout << "Sending desired export of: " << t_watts << endl;
        Message reply(bus);
        MsgArg arg("d", t_watts);
        QStatus status = proxy.MethodCall(INTF_NAME, "ExportPower", &arg, 1, 0);
        return status;
    }

    QStatus SignalPJM(double tValue){
        Message reply(bus);
        MsgArg arg("d", tValue);
        QStatus status = proxy.MethodCall(INTF_NAME, "SignalPJM", &arg, 1, 0);
        return status;
    } // end signal pjm

    void pathDelim(){
        vector <string> path, temp;
        int num;
        double val;
        path = tools.stringDelim(proxy.GetPath().c_str(),'/');
        num = path.size();
        for (int i = 0; i < num; i++){
            if (path[i].find("region") == 0){
                temp = tools.stringDelim(path[i],'_');
                val = stod(temp[1]);
                properties.insert(pair<string,double>("region",val));
            } else if (path[i].find("feeder") == 0){
                temp = tools.stringDelim(path[i],'_');
                val = stod(temp[1]);
                properties.insert(pair<string,double>("feeder",val));
            }
        } // end string comparison
    } // END PATH DELIM

    void telemetry() {
        properties.clear();
        //cout << "Polling asset telemetry " << endl;
        MsgArg dict;

        QStatus status = proxy.GetAllProperties(INTF_NAME, dict);
        if (ER_OK == status){
            pathDelim();
            MsgArg * entries = NULL;
            size_t num = 0;
            dict.Get("a{sv}", &num, &entries);
            for (size_t i = 0; i < num; ++i) {
                char * key;
                double val;
                status = entries[i].Get("{sd}", &key, &val);
                if (ER_OK == status) {
                    properties.insert(pair<string,double>(key,val));
                }
            }
        }
    } // END TELEMETRY

    vector<double> GetProperties() {
        vector<double> values;
        values.reserve(properties.size());
        for(auto it = properties.cbegin(); it != properties.cend(); ++it)
        {
            values.emplace_back(it->second);
        }

        return values;
```

```
    } // END PRINT TELEMETRY
};

static void Help()
{
    cout << "q                    quit" << endl;
    cout << "l                    list all discovered Assets" << endl;
    cout << "t                    print telemetry" << endl;
    cout << "i <int watts>        import power signal" << endl;
    cout << "e <int watts>        export power signal" << endl;
    cout << "x <int watts>        import power test" << endl;
    cout << "y <int watts>        export power test" << endl;
    cout << "h                    display this help message" << endl;
}

unsigned int ListAssets(BusAttachment& bus, Observer* observer)
{
    unsigned int ctr = 0;
    ProxyBusObject proxy = observer->GetFirst();
    for (; proxy.IsValid(); proxy = observer->GetNext(proxy)) {
        ctr++;
        AssetProxy asset(proxy, bus);
    } // for
    cout << ctr << endl;
    return  ctr;
}

vector<vector<double>> Telemetry(BusAttachment& bus, Observer* observer)
{
    unsigned int count = ListAssets(bus, observer);
    vector<vector<double>> assets;
    assets.reserve(count);
    vector<double> data;
    ProxyBusObject proxy = observer->GetFirst();
    for (; proxy.IsValid(); proxy = observer->GetNext(proxy))
    {
        AssetProxy asset(proxy, bus);
        asset.telemetry();
        data = asset.GetProperties();
        assets.emplace_back(data);
    } //for
    string file = "save_";
    file.append(tools.GetTime());
    tools.StoreData(assets, file);
    return assets;
} // END TELEMETRY

static void CallImportPower(BusAttachment& bus, Observer* observer, double t_watts)
{
    ProxyBusObject proxy = observer->GetFirst();
    for (; proxy.IsValid(); proxy = observer->GetNext(proxy)) {
        AssetProxy asset(proxy, bus);
        QStatus status = asset.ImportPower(t_watts);
        if (ER_OK != status) {
            cerr << "Could not set desired import power " << proxy.GetUniqueName() << ":"
             ↪  << proxy.GetPath() << endl;
            continue;
        }
    }
}

static void TestImport(BusAttachment& bus, Observer* observer, double t_watts)
{
    QStatus status;
    ProxyBusObject proxy = observer->GetFirst();
    for (; proxy.IsValid(); proxy = observer->GetNext(proxy)) {
        if (!strcmp(proxy.GetPath().c_str(),"/asset_bess/region_1/feeder_1")){
            AssetProxy asset(proxy, bus);
            asset.telemetry();
            while (asset.properties["import_energy"] >= 0){
                cout << "The import energy is: " << asset.properties["import_energy"] <<
                 ↪  endl;
                status = asset.ImportPower(t_watts);
                if (ER_OK != status) {
                    cerr << "Could not set desired import power " << proxy.GetUniqueName()
                     ↪  << ":" << proxy.GetPath() << endl;
                    continue;
                }
                // wait one hour and then tell the system to stop with a zero signal.
                this_thread::sleep_for(chrono::seconds(10));
                status = asset.ImportPower(0);
                if (ER_OK != status) {
                    cerr << "Could not set desired import power " << proxy.GetUniqueName()
                     ↪  << ":" << proxy.GetPath() << endl;
                    continue;
                }
                asset.telemetry();
            }
        }
    }
    cout <<"/n/n/n****TEST COMPLETE****/n/n/n";
```

```
}
static void TestExport(BusAttachment& bus, Observer* observer, double t_watts)
{
    QStatus status;
    ProxyBusObject proxy = observer->GetFirst();
    for (; proxy.IsValid(); proxy = observer->GetNext(proxy)) {
        if (!strcmp(proxy.GetPath().c_str(),"/asset_bess/region_1/feeder_1")){
            AssetProxy asset(proxy, bus);
            asset.telemetry();
            while (asset.properties["export_energy"] >= 0){
                cout << "The export energy is: " << asset.properties["export_energy"] <<
                  ↪  endl;
                status = asset.ExportPower(t_watts);
                if (ER_OK != status) {
                    cerr << "Could not set desired export power " << proxy.GetUniqueName()
                      ↪  << ":" << proxy.GetPath() << endl;
                    continue;
                }
                // wait one hour and then tell the system to stop with a zero signal.
                this_thread::sleep_for(chrono::seconds(10));

                status = asset.ExportPower(0);
                if (ER_OK != status) {
                    cerr << "Could not set desired export power " << proxy.GetUniqueName()
                      ↪  << ":" << proxy.GetPath() << endl;
                    continue;
                }
                asset.telemetry();
            }
        }
    }
    cout <<"/n/n/n****TEST COMPLETE****/n/n/n";
}

static void CallExportPower(BusAttachment& bus, Observer* observer, double t_watts)
{
    ProxyBusObject proxy = observer->GetFirst();
    for (; proxy.IsValid(); proxy = observer->GetNext(proxy)) {
        AssetProxy asset(proxy, bus);
        QStatus status = asset.ExportPower(t_watts);
        if (ER_OK != status) {
            cerr << "Could not set desired export power " << proxy.GetUniqueName() << ":"
              ↪  << proxy.GetPath() << endl;
            continue;
        }
    }
}

static void RegD(BusAttachment& bus, Observer* observer, double tValue)
{
    ProxyBusObject proxy = observer->GetFirst();
    for (; proxy.IsValid(); proxy = observer->GetNext(proxy))
    {
        AssetProxy asset(proxy, bus);
        QStatus status = asset.SignalPJM(tValue);
        if (ER_OK != status){
            cerr << "Could not send RegD to: " << proxy.GetUniqueName() << endl;
           continue;
        } // if

    } // for

} // end RegD

static void PJMControl(BusAttachment& bus, Observer* observer)
{
    //call collect data function
    vector<vector<double>> data = Telemetry(bus, observer);
    //call aggregator
    vector<double> totals = tools.Aggregate(data);

    string file = "samplePJM.txt";
    vector<double> schedule = tools.ImportSchedule(file);

    for(unsigned int i=0; i<schedule.size(); i++)
    {
            auto startTime = chrono::high_resolution_clock::now();
        //send signal function
        RegD(bus, observer, schedule[i]);
        //emulate models
        auto endTime = chrono::high_resolution_clock::now();
            chrono::duration<double, milli> elapsed = startTime - endTime;
        int wait = 2000 - elapsed.count();                // 2 seconds - processing time
           this_thread::sleep_for(chrono::milliseconds(wait));
    } //for

    //call collect data function
    data = Telemetry(bus, observer);
} // END PJM CONTROL


static bool Parse(BusAttachment& bus, Observer* observer, const string & input)
{
    char cmd;
    vector <string> tokens;
```

```
    string option;

    if (input == "") {
        return true;
    }

    stringstream s(input);
    while(!s.eof()) {
        string tmp;
        s >> tmp;
        tokens.push_back(tmp);
    }

    if (tokens.empty()) {
        return true;
    }

    cmd = input[0];

    switch (cmd) {
    case 'q':
        return false;

    case 'l':
        ListAssets(bus, observer);
        break;

    case 't':
        Telemetry(bus, observer);
        break;

    case 'i':
        if (tokens.size() < 2) {
            Help();
            break;
        }
        option = tokens.at(1);
        CallImportPower(bus, observer,stod(option));
        break;

    case 'e':
        if (tokens.size() < 2) {
            Help();
            break;
        }
        option = tokens.at(1);
        CallExportPower(bus, observer,stod(option));
        break;

    case 'x':
        if (tokens.size() < 2) {
            Help();
            break;
        }
        option = tokens.at(1);
        TestImport(bus, observer,stod(option));
        break;

    case 'y':
        if (tokens.size() < 2) {
            Help();
            break;
        }
        option = tokens.at(1);
        TestExport(bus, observer,stod(option));
        break;

    case 's':
        PJMControl(bus, observer);
        break;

    case 'h':
    default:
        Help();
        break;
    }

    return true;
}

static QStatus BuildInterface(BusAttachment& bus)
{
    QStatus status;

    InterfaceDescription* intf = NULL;
    status = bus.CreateInterface(INTF_NAME, intf);
    QCC_ASSERT(ER_OK == status);

    status = intf->AddMethod("ImportPower", "d", NULL, "watts", MEMBER_ANNOTATE_NO_REPLY);
    assert (ER_OK == status);

    status = intf->AddMethod("ExportPower", "d", NULL, "watts", MEMBER_ANNOTATE_NO_REPLY);
    assert (ER_OK == status);

    status = intf->AddMethod("SignalPJM", "d", NULL, "RegD", MEMBER_ANNOTATE_NO_REPLY);
    assert (ER_OK == status);

    status = intf->AddProperty("import_power", "d", PROP_ACCESS_READ);
    QCC_ASSERT(ER_OK == status);
```

```
    status = intf->AddPropertyAnnotation("import_power",
      ↪   "org.freedesktop.DBus.Property.EmitsChangedSignal", "false");
    QCC_ASSERT(ER_OK == status);

    status = intf->AddProperty("export_power", "d", PROP_ACCESS_READ);
    QCC_ASSERT(ER_OK == status);

    status = intf->AddPropertyAnnotation("export_power",
      ↪   "org.freedesktop.DBus.Property.EmitsChangedSignal", "false");
    QCC_ASSERT(ER_OK == status);

    status = intf->AddProperty("import_energy", "d", PROP_ACCESS_READ);
    QCC_ASSERT(ER_OK == status);

    status = intf->AddPropertyAnnotation("import_energy",
      ↪   "org.freedesktop.DBus.Property.EmitsChangedSignal", "false");
    QCC_ASSERT(ER_OK == status);

    status = intf->AddProperty("export_energy", "d", PROP_ACCESS_READ);
    QCC_ASSERT(ER_OK == status);

    status = intf->AddPropertyAnnotation("export_energy",
      ↪   "org.freedesktop.DBus.Property.EmitsChangedSignal", "false");
    QCC_ASSERT(ER_OK == status);

    status = intf->AddProperty("idle_characteristic", "d", PROP_ACCESS_READ);
    QCC_ASSERT(ER_OK == status);

    status = intf->AddPropertyAnnotation("idle_characteristic",
      ↪   "org.freedesktop.DBus.Property.EmitsChangedSignal", "false");
    QCC_ASSERT(ER_OK == status);

    status = intf->AddProperty("import_characteristic", "d", PROP_ACCESS_READ);
    QCC_ASSERT(ER_OK == status);

    status = intf->AddPropertyAnnotation("import_characteristic",
      ↪   "org.freedesktop.DBus.Property.EmitsChangedSignal", "false");
    QCC_ASSERT(ER_OK == status);

    status = intf->AddProperty("export_characteristic", "d", PROP_ACCESS_READ);
    QCC_ASSERT(ER_OK == status);

    status = intf->AddPropertyAnnotation("export_characteristic",
      ↪   "org.freedesktop.DBus.Property.EmitsChangedSignal", "false");
    QCC_ASSERT(ER_OK == status);

    status = intf->AddProperty("time", "d", PROP_ACCESS_READ);
    QCC_ASSERT(ER_OK == status);

    status = intf->AddPropertyAnnotation("time",
      ↪   "org.freedesktop.DBus.Property.EmitsChangedSignal", "false");
    QCC_ASSERT(ER_OK == status);

    intf->Activate();

    return status;
}

static QStatus SetupBusAttachment(BusAttachment& bus)
{
    QStatus status;
    status = bus.Start();
    QCC_ASSERT(ER_OK == status);
    status = bus.Connect();

    if (ER_OK != status) {
        return status;
    }

    status = BuildInterface(bus);
    QCC_ASSERT(ER_OK == status);

    return status;
}

class AssetListener :
    public MessageReceiver,
    public Observer::Listener {
    static const char * props[];
  public:
    Observer* observer;
    BusAttachment* bus;

    virtual void ObjectDiscovered(ProxyBusObject& proxy) {
        cout << "[listener] Asset " << proxy.GetUniqueName() << ":"
             << proxy.GetPath() << " has just been discovered." << endl;
    }

    virtual void ObjectLost(ProxyBusObject& proxy) {
        cout << "[listener] Asset " << proxy.GetUniqueName() << ":"
             << proxy.GetPath() << " no longer exists." << endl;
    }
};

const char* AssetListener::props[] = {
    "IsOpen"
};

int CDECL_CALL main(int argc, char** argv)
```

```
{
    QCC_UNUSED(argc);
    QCC_UNUSED(argv);

    if (AllJoynInit() != ER_OK) {
        return EXIT_FAILURE;
    }
#ifdef ROUTER
    if (AllJoynRouterInit() != ER_OK) {
        AllJoynShutdown();
        return EXIT_FAILURE;
    }
#endif

    BusAttachment* bus = new BusAttachment("Asset_consumer", true);

    if (ER_OK != SetupBusAttachment(*bus)) {
        return EXIT_FAILURE;
    }

    const char* intfname = INTF_NAME;
    Observer* obs = new Observer(*bus, &intfname, 1);
    AssetListener* listener = new AssetListener();
    listener->observer = obs;
    listener->bus = bus;
    obs->RegisterListener(*listener);

    bool done = false;
    printf("\n\n\n**********\tEMS\t**********\n\n\n");
    Help();
    printf("\n\n\n");
    while (!done) {
        string input;
        cout << "> ";
        getline(cin, input);
        done = !Parse(*bus, obs, input);
    }

    // Cleanup
    obs->UnregisterAllListeners();
    delete obs; // Must happen before deleting the original bus
    delete listener;
    delete bus;
    bus = NULL;

#ifdef ROUTER
    AllJoynRouterShutdown();
#endif
    AllJoynShutdown();
    return EXIT_SUCCESS;
}
```

A.2.2 utility.cpp

```
#include "utility.h"
using namespace std;

string Utility::GetTime()
{
    chrono::time_point<chrono::system_clock> time_now = chrono::system_clock::now();
    time_t time_now_t = chrono::system_clock::to_time_t(time_now);
    tm now_tm = *localtime(&time_now_t);
    ostringstream ss;
    ss << put_time(&now_tm, "%Y%m%d_%H_%M_%S");
    return ss.str();
} // END GET TIME

map<string, string> Utility::setConfig(string t_fileName)
{
    map<string, string> configMap;
    string line;
    string key;
    string value;
    ifstream config(t_fileName);

    if (config.is_open()){
        while (getline(config,line)){
            if (line.front() == '#') continue;
            istringstream is_line(line);
            if (getline(is_line,key,'=')){
                if (getline(is_line, value)){
                    configMap.insert(pair<string, string>(key,value));
                } //END GET VALUE
            } //END GET LINES
        } //END WHILE LINE
    } else {
        cout << "Unable to open file " << t_fileName << endl;
    }//END IF/ELSE FILE IPEN
```

```
    return configMap;
} //END SET CONFIG

vector<vector<string>> Utility::readCSV(string t_fileName)
{
    string line;
    string cell;
    vector<vector<string>> csvMap;

    ifstream file(t_fileName);
    if (file.is_open()){
        cout << "\tInitializing modBus register map from " << t_fileName << endl;
        while (getline(file,line)){
            vector<string> csvRow;
            istringstream s(line);
            while (getline(s,cell,',')){
                csvRow.push_back(cell);
            } //END GET COLUMN
            csvMap.push_back(csvRow);
        } //END GET ROW
    file.close();
    } else {
        cerr << "Unable to open register initialization file" << t_fileName << endl;
    } //END IF/ELSE OPEN
    return csvMap;
} //END READ CSV

void Utility::writeCSV(vector<vector<string>> t_data, string t_fileName)
{
    int ncol = t_data[0].size();
    int nrow = t_data.size();

    ofstream file(t_fileName.c_str());
    if (file.is_open()){
        cout << "\tWriting to file: " << t_fileName << endl;
        for(int i = 0; i < nrow; i++){
            for(int j = 0; j < ncol; j++){
                file << t_data[i][j] << ", ";
            }
            file << endl;
        }

        file.close();
    } else {
        cout << "Unable to open file" << endl;
    }

} //END WRITE CSV

void Utility::StoreData(vector<vector<double>> tData, string tFileName)
{
    int ncol = tData[0].size();
    int nrow = tData.size();

    ofstream file(tFileName.c_str());
    if(file.is_open())
    {

        for(int i=0; i<nrow; i++)
        {

            for(int j=0; j<ncol; j++)
            {
                file << tData[i][j] << ", ";
            } // for

            file << endl;
        } // for

    file.close();
    } else {
        cout << "Unable to open file\n";
    } // if/else
}


vector<vector<string>> Utility::filterData(vector<vector<string>> t_data, string t_header,
  ↪  string t_criterion)
{
    vector<vector<string>> filter;
    vector<string> header = t_data[0];
    int nrow = t_data.size();

    ptrdiff_t col = find(header.begin(), header.end(), t_header) - header.begin();
    for(int i = 1; i < nrow; i++){
        if (t_data[i][col] == t_criterion){
            filter.push_back(t_data[i]);
        }
    }
    return filter;
} //END FILTER DATA

void Utility::writeText(string t_fileName, string t_message)
{
    ofstream file(t_fileName,ios::app);
```

```
    if (file.is_open())
    {
        file << t_message << "\n";
        file.close();
    } else {
        cout << "Unable to open file" << endl;
    }
} //END WRITE TEXT

string Utility::getDateTime()
{
    time_t rawtime;
    struct tm * timeinfo;
    char buffer[80];

    time (&rawtime);
    timeinfo = localtime(&rawtime);

    strftime(buffer,sizeof(buffer),"%d-%b-%Y %H:%M:%S",timeinfo);
    return buffer;
} //END GET DATE/TIME

vector<string> Utility::stringDelim(string t_string, char t_delim)
{
    string temp;
    vector<string> deliminated;

    istringstream s(t_string);
    while (getline(s,temp, t_delim)){
        deliminated.push_back(temp);
    } //END GET COLUMN

    return deliminated;
} //END READ CSV


vector<double> Utility::Aggregate(vector<vector<double>> tData)
{
    int iMax = tData.size();
    int jMax = tData[0].size();
    vector<double> totals;
    totals.reserve(jMax);

    double sum;
    for(int j=0; j<jMax; j++)
    {
        sum = 0;
        for(int i=0; i<iMax; i++)
        {
            sum = sum + tData[i][j];
        } // for

        totals.emplace_back(sum);
    } // for

    return totals;
} // END AGGREGATE
```

```
vector<double> Utility::ImportSchedule(string tFileName)
{
    string line;
    vector<double> schedule;

    ifstream file(tFileName.c_str());
    if (file.is_open()){

        while (getline(file,line)){
            schedule.push_back(stod(line));
        } //while

    file.close();
    } else {
        cerr << "Unable to open register initialization file" << tFileName << endl;
    } //if/else

    return schedule;
} //END IMPORT SCHEDULE
```

A.2.3 utility.h

```
#ifndef UTILITY_H
#define UTILITY_H
#include <algorithm>
#include <iostream>
#include <iomanip>
#include <string>
#include <vector>
#include <map>
#include <fstream>
#include <sstream>
#include <chrono>
#include <ctime>

using std::vector;
using std::string;

class Utility
{
    public:
        vector<vector<string>> readCSV(string t_fileName);
        void writeCSV(vector<vector<string>> t_data, string t_fileName);
        vector<vector<string>>  filterData(vector<vector<string>> t_data, string t_header,
          ↪   string t_criterion);
        void writeText(string t_fileName, string t_message);
        string getDateTime();
        std::map<string, string> setConfig(string t_fileName);
        vector<string> stringDelim(string t_string, char t_delim);
        vector<double> Aggregate(vector<vector<double>> tData);
        vector<double> ImportSchedule(string tFileName);
        void StoreData(vector<vector<double>> tData, string tFileName);
        string GetTime();
    protected:
    private:
};

#endif // UTILITY_H
```

A.2.4 Makefile

```
# Copyright AllSeen Alliance. All rights reserved.
#
#    Permission to use, copy, modify, and/or distribute this software for any
#    purpose with or without fee is hereby granted, provided that the above
#    copyright notice and this permission notice appear in all copies.
#
#    THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
#    WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
#    MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
#    ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
#    WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
#    ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
#    OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

# Modified by V-SQUARED, June 2016

CPPOPTS :=

ALLJOYN_DIST := $(AJ_ROOT)/build/$(OS)/$(CPU)/debug/dist/cpp

OBJ_DIR := obj/Debug

BIN_DIR := bin/Debug

ALLJOYN_CPPLIB := $(ALLJOYN_DIST)/lib
COMMON_INC := $(AJ_ROOT)/common/inc

CXXFLAGS = -Wall -pipe -std=c++11 -fno-rtti -fno-exceptions -Wno-long-long -Wno-deprecated
  ↪   -g -DQCC_OS_LINUX -DQCC_OS_GROUP_POSIX -DQCC_DBG -O0 -Wno-write-strings

LIBS = -lstdc++ -lpthread -lalljoyn -lajrouter -lrt -lm

default: EMS

EMS: ems.o utility.o
        mkdir -p $(BIN_DIR)
        $(CXX) -o $(BIN_DIR)/EMS $(OBJ_DIR)/ems.o $(OBJ_DIR)/utility.o -L$(ALLJOYN_CPPLIB)
  ↪   $(LIBS)

ems.o: ems.cpp utility.h $(ALLJOYN_LIB)
        mkdir -p $(OBJ_DIR)
        $(CXX) -c $(CXXFLAGS) -I$(ALLJOYN_DIST)/inc -o $(OBJ_DIR)/$@ ems.cpp

utility.o: utility.cpp utility.h
        mkdir -p $(OBJ_DIR)
        $(CXX) -c $(CXXFLAGS) -o $(OBJ_DIR)/$@ utility.cpp

all_clean: clean
        rmdir $(OBJ_DIR)
        rmdir $(BIN_DIR)
```

```
	rmdir bin
	rmdir obj
	rm *.plt
clean:
	rm -f $(BIN_DIR)/EMS
	rm -f $(OBJ_DIR)/ems.o
	rm -f $(OBJ_DIR)/utility.o
```

A.3 Distributed Management System

A.3.1 main.cpp

```
/******************************************************************************
 *    Copyright (c) Open Connectivity Foundation (OCF), AllJoyn Open Source
 *    Project (AJOSP) Contributors and others.
 *
 *    SPDX-License-Identifier: Apache-2.0
 *
 *    All rights reserved. This program and the accompanying materials are
 *    made available under the terms of the Apache License, Version 2.0
 *    which accompanies this distribution, and is available at
 *    http://www.apache.org/licenses/LICENSE-2.0
 *
 *    Copyright (c) Open Connectivity Foundation and Contributors to AllSeen
 *    Alliance. All rights reserved.
 *
 *    Permission to use, copy, modify, and/or distribute this software for
 *    any purpose with or without fee is hereby granted, provided that the
 *    above copyright notice and this permission notice appear in all
 *    copies.
 *
 *    THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL
 *    WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED
 *    WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE
 *    AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL
 *    DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR
 *    PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER
 *    TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
 *    PERFORMANCE OF THIS SOFTWARE.
******************************************************************************/

#include <cstdio>
#include <iostream>
#include <vector>
#include <memory>
#include <stdlib.h>
#include <errno.h>
#include <string>
#include <sstream>
#include <chrono>
#include <ctime>
#include <thread>

#include <alljoyn/Status.h>
#include <alljoyn/AboutObj.h>
#include <alljoyn/BusAttachment.h>
#include <alljoyn/Init.h>

#include "der.h"

using namespace std;
using namespace ajn;

/*constants*/
static const char * INTF_NAME = "edu.pdx.powerlab.demo";
static qcc::String s_advertisedName = INTF_NAME;
bool done = false;

/******************************************************************************
 *      Session Port Listener
 *      ---------------------
 *      AllJoyn class to join a session with the EMS
******************************************************************************/
class SPL : public SessionPortListener {
    virtual bool AcceptSessionJoiner(SessionPort sessionPort, const char* joiner, const
     ↪  SessionOpts& opts) {
        QCC_UNUSED(sessionPort);
        QCC_UNUSED(joiner);
        QCC_UNUSED(opts);
        return true;
    }
};

SPL g_session_port_listener;
```

```
/*******************************************************************************
 *      Build Interface
 *      ---------------
 *      AllJoyn function to establish the Methods/Properties/Signals that will
 *      be implemented by the BusObject.
 *
 *      @Method(ImportPower): A control signal from the EMS with a desired watt
 *          value that needs to be consumed by the asset for an hour.
 *      @Method(ExportPower): A control signal fromt he EMS with a desired watt
 *          value that needs to be generated/shed by the asset for an hour.
 *      @Property(import_power): The assets rated input power capacity in watts.
 *      @Property(export_power): The assets rated generation/shed power capacity
 *          in watts.
 *      @Property(import_energy): The assets total available input power capacity
 *          in watt-hours.
 *      @Property(export_energy): The assets total available generation/shed power
 *          capacity in watt-hours.
 *      @Property(import_characteristic): The assets input power response approximated
 *          as a linear slope.
 *      @Property(export_characteristic): The assets generation/shed response
 *          approximated as a linear slope.
 *      @Property(time): A timestampe used to coordinate the aggregated assets.
 *******************************************************************************/
static QStatus BuildInterface(BusAttachment& bus)
{
    QStatus status;

    InterfaceDescription* intf = NULL;
    status = bus.CreateInterface(INTF_NAME, intf);
    QCC_ASSERT(ER_OK == status);

    status = intf->AddMethod("ImportPower", "d", NULL, "watts", MEMBER_ANNOTATE_NO_REPLY);
    assert (ER_OK == status);

    status = intf->AddMethod("ExportPower", "d", NULL, "watts", MEMBER_ANNOTATE_NO_REPLY);
    assert (ER_OK == status);

    status = intf->AddMethod("SignalPJM", "d", NULL, "RegD", MEMBER_ANNOTATE_NO_REPLY);
    assert (ER_OK == status);

    status = intf->AddProperty("import_power", "d", PROP_ACCESS_READ);
    QCC_ASSERT(ER_OK == status);

    status = intf->AddPropertyAnnotation("import_power",
      ↪   "org.freedesktop.DBus.Property.EmitsChangedSignal", "false");
    QCC_ASSERT(ER_OK == status);

    status = intf->AddProperty("export_power", "d", PROP_ACCESS_READ);
    QCC_ASSERT(ER_OK == status);

    status = intf->AddPropertyAnnotation("export_power",
      ↪   "org.freedesktop.DBus.Property.EmitsChangedSignal", "false");
    QCC_ASSERT(ER_OK == status);

    status = intf->AddProperty("import_energy", "d", PROP_ACCESS_READ);
    QCC_ASSERT(ER_OK == status);

    status = intf->AddPropertyAnnotation("import_energy",
      ↪   "org.freedesktop.DBus.Property.EmitsChangedSignal", "false");
    QCC_ASSERT(ER_OK == status);

    status = intf->AddProperty("export_energy", "d", PROP_ACCESS_READ);
    QCC_ASSERT(ER_OK == status);

    status = intf->AddPropertyAnnotation("export_energy",
      ↪   "org.freedesktop.DBus.Property.EmitsChangedSignal", "false");
    QCC_ASSERT(ER_OK == status);

    status = intf->AddProperty("idle_characteristic", "d", PROP_ACCESS_READ);
    QCC_ASSERT(ER_OK == status);

    status = intf->AddPropertyAnnotation("idle_characteristic",
      ↪   "org.freedesktop.DBus.Property.EmitsChangedSignal", "false");
    QCC_ASSERT(ER_OK == status);

    status = intf->AddProperty("import_characteristic", "d", PROP_ACCESS_READ);
    QCC_ASSERT(ER_OK == status);

    status = intf->AddPropertyAnnotation("import_characteristic",
      ↪   "org.freedesktop.DBus.Property.EmitsChangedSignal", "false");
    QCC_ASSERT(ER_OK == status);

    status = intf->AddProperty("export_characteristic", "d", PROP_ACCESS_READ);
    QCC_ASSERT(ER_OK == status);

    status = intf->AddPropertyAnnotation("export_characteristic",
      ↪   "org.freedesktop.DBus.Property.EmitsChangedSignal", "false");
    QCC_ASSERT(ER_OK == status);

    status = intf->AddProperty("time", "d", PROP_ACCESS_READ);
    QCC_ASSERT(ER_OK == status);

    status = intf->AddPropertyAnnotation("time",
      ↪   "org.freedesktop.DBus.Property.EmitsChangedSignal", "false");
    QCC_ASSERT(ER_OK == status);

    intf->Activate();

    return status;
```

```
}


/***************************************************************************
 *      Setup Bus Attachment
 *      --------------------
 *      AllJoyn function to connect to a DBus session and populate the about
 *      properties of the asset.
 ***************************************************************************/
static QStatus SetupBusAttachment(BusAttachment& bus, AboutData& aboutData, SessionPort
 ↪   SERVICE_PORT)
{
    QStatus status;
    status = bus.Start();
    QCC_ASSERT(ER_OK == status);
    status = bus.Connect();
    if (status != ER_OK) {
        return status;
    }

    status = BuildInterface(bus);
    QCC_ASSERT(ER_OK == status);

    SessionOpts opts(SessionOpts::TRAFFIC_MESSAGES, true, SessionOpts::PROXIMITY_ANY,
      ↪   TRANSPORT_ANY);
    bus.BindSessionPort(SERVICE_PORT, opts, g_session_port_listener);

    /* set up totally uninteresting about data */
    //AppId is a 128bit uuid
    uint8_t appId[] = { 0x01, 0xB3, 0xBA, 0x14,
                        0x1E, 0x82, 0x11, 0xE4,
                        0x86, 0x51, 0xD1, 0x56,
                        0x1D, 0x5D, 0x46, 0xB0 };
    aboutData.SetAppId(appId, 16);
    aboutData.SetDeviceName("Foobar 2000 Door Security");
    //DeviceId is a string encoded 128bit UUID
    aboutData.SetDeviceId("93c06771-c725-48c2-b1ff-6a2a59d445b8");
    aboutData.SetAppName("Application");
    aboutData.SetManufacturer("Manufacturer");
    aboutData.SetModelNumber("123456");
    aboutData.SetDescription("A poetic description of this application");
    aboutData.SetDateOfManufacture("2014-03-24");
    aboutData.SetSoftwareVersion("0.1.2");
    aboutData.SetHardwareVersion("0.0.1");
    aboutData.SetSupportUrl("http://www.example.org");
    if (!aboutData.IsValid()) {
        cerr << "Invalid about data." << endl;
        return ER_FAIL;
    }
    return status;
}


/***************************************************************************
 *      Asset (BusObject, Asset)
 *      ------------------------
 *      The asset is the BusObject as well as asset. Therefore it inherits
 *      each of the parent class functions.
 *
 *      BusObject(Get, ImportPowerHandler, ExportPowerHandler)
 *
 *      Asset(State, Mode, Properties, Logging)
 *          @State : Checks the health of the asset to determine its Mode.
 *              i.e.(bypass, backup, nominal, disconnected)
 *          @Mode : The asset Mode dictates what it can participate in.
 *          @Properties : The asset polls its subsystems to determine the BusObject
 *              properties.
 *          @Logging : The system stores error/data locally for system analysis
 *              and troubleshooting.
 ***************************************************************************/
class Asset : public BusObject, public DER {
  private:
    BusAttachment& bus;

  public:
    Asset(BusAttachment& bus, const char * path)
        : BusObject(path),
        bus(bus)
    {
        const InterfaceDescription* intf = bus.GetInterface(INTF_NAME);
        QCC_ASSERT(intf);
        AddInterface(*intf, ANNOUNCED);

        /** Register the method handlers with the object **/
        const MethodEntry methodEntries[] = {
            { intf->GetMember("ImportPower"),
              ↪   static_cast<MessageReceiver::MethodHandler>(&Asset::ImportPowerHandler)
              ↪   },
            { intf->GetMember("ExportPower"),
              ↪   static_cast<MessageReceiver::MethodHandler>(&Asset::ExportPowerHandler)
              ↪   },
            { intf->GetMember("SignalPJM"),
              ↪   static_cast<MessageReceiver::MethodHandler>(&Asset::PJMHandler) },
        };
```

```
        QStatus status = AddMethodHandlers(methodEntries, sizeof(methodEntries) /
          ↪   sizeof(methodEntries[0]));
        if (ER_OK != status) {
            cerr << "Failed to register method handlers for Asset." << endl;
        }
    }

    ~Asset()
    {
    }

    /* property getters */
    QStatus Get(const char*ifcName, const char*propName, MsgArg& val)
    {
        if (strcmp(ifcName, INTF_NAME)) {
            return ER_FAIL;
        }

        if (!strcmp(propName, "import_power")) {
            val.Set("d", m_importWatts);
        } else if (!strcmp(propName, "export_power")) {
            val.Set("d", m_exportWatts);
        } else if (!strcmp(propName, "import_energy")) {
            val.Set("d", m_importWattHours);
        } else if (!strcmp(propName, "export_energy")) {
            val.Set("d", m_exportWattHours);
        } else if (!strcmp(propName, "idle_characteristic")) {
            val.Set("d", m_idleChar);
        } else if (!strcmp(propName, "import_characteristic")) {
            val.Set("d", m_importChar);
        } else if (!strcmp(propName, "export_characteristic")) {
            val.Set("d", m_exportChar);
        } else if (!strcmp(propName, "time")) {
            double t = static_cast<unsigned long int> (time(NULL));
            val.Set("d", t);
        } else {
            return ER_FAIL;
        }
        return ER_OK;
    }

    void ImportPowerHandler(const InterfaceDescription::Member * t_member, Message &
      ↪   t_msg)
    {
        QCC_UNUSED(t_member);
        m_exportControl = 0;
        m_importControl = t_msg->GetArg(0)->v_double;
        printf("\t***Import Power Method recieved with a desired setting of %g.\n\n\n",
          ↪   m_importControl);
                GetTime();
                ImportPower();

    } //END IMPORT POWER HANDLER

    void ExportPowerHandler(const InterfaceDescription::Member * t_member, Message &
      ↪   t_msg)
    {
        QCC_UNUSED(t_member);
        m_importControl = 0;
        m_exportControl = t_msg->GetArg(0)->v_double;
        printf("\t***Export Power Method recieved with a desired setting of %g.\n\n\n",
          ↪   m_exportControl);
                GetTime();
                ExportPower();

    } //END EXPORT POWER HANDLER

    void PJMHandler(const InterfaceDescription::Member *t_member, Message &t_msg)
    {
        QCC_UNUSED(t_member);
        double m_regulate = t_msg->GetArg(0)->v_double;
        Regulate();
        ImportPower();
        ExportPower();
        Losses();
    } // END PJM HANDLER
};


/******************************************************************************
 *      Help
 *      ----
 *      Display input criteria for Command Line Interface.
******************************************************************************/
static void Help()
{
    cout << "q          quit" << endl;
    cout << "h          show this help message" << endl;
}


/******************************************************************************
 *      Parse
 *      -----
 *      Interprete input from the Command Line.
```

```
*******************************************************************************/
static bool Parse(const string & input)
{
    char cmd;
    vector <string> tokens;
    string option;

    if (input == "") {
        return true;
    }

    stringstream s(input);
    while(!s.eof()) {
        string tmp;
        s >> tmp;
        tokens.push_back(tmp);
    }

    if (tokens.empty()) {
        return true;
    }

    cmd = input[0];

    switch (cmd) {
    case 'q':
        return false;

    case 'h':
    default:
        Help();
        break;
    }

    return true;
}


/*******************************************************************************
 *      Shutdown
 *      --------
 *      Stop the AllJoyn Bundled Router and AllJoyn threads.
*******************************************************************************/
static void Shutdown() {
    #ifdef ROUTER
        AllJoynRouterShutdown();
    #endif
        AllJoynShutdown();
}

/*******************************************************************************
 *      Main
 *      ----
 *      Alljoyn_thread
 *      Asset_thread
*******************************************************************************/
int CDECL_CALL main(int argc, char** argv)
{
    if (AllJoynInit() != ER_OK) {
        return EXIT_FAILURE;
    }
#ifdef ROUTER
    if (AllJoynRouterInit() != ER_OK) {
        AllJoynShutdown();
        return EXIT_FAILURE;
    }
#endif
    int assignment = atoi(argv[1]);
    SessionPort SERVICE_PORT = assignment;
    cout << SERVICE_PORT << endl;

    BusAttachment* bus = NULL;
    bus = new BusAttachment("Asset_provider", true);
    QCC_ASSERT(bus != NULL);
    AboutData aboutData("en");
    AboutObj* aboutObj = new AboutObj(*bus);
    QCC_ASSERT(aboutObj != NULL);

    if (ER_OK != SetupBusAttachment(*bus, aboutData, SERVICE_PORT)) {
        delete aboutObj;
        aboutObj = NULL;
        delete bus;
        bus = NULL;
        return EXIT_FAILURE;
    }
    string tempPath = "/asset_bess/region_1/feeder_" + to_string(assignment);
    const char * SERVICE_PATH = tempPath.c_str();
    Asset * asset = new Asset(*bus, SERVICE_PATH);

    if (ER_OK != bus->RegisterBusObject(*asset)) {
        delete asset;
    }

    aboutObj->Announce(SERVICE_PORT, aboutData);

    printf("\n\n\n**********\tDMS\t**********\n\n\n");
    Help();
    printf("\n\n\n");
```

```
    while (!done) {
       string input;
       cout << "> ";
       getline(cin, input);
       done = !Parse(input);
    }

    delete asset;
    asset = NULL;

    delete aboutObj;
    aboutObj = NULL;

    delete bus;
    bus = NULL;

    Shutdown();
    return EXIT_SUCCESS;
}
```

A.3.2 bess.cpp

```
#include "bess.h"

DER::DER() :
    m_importControl(0),
    m_exportControl(0),
    m_importWatts(0),
    m_exportWatts(0),
    m_importWattHours(1000),
    m_idleChar(1),
    m_importChar(1000),
    m_exportChar(1000)
{
    //ctor
    DER::Random();
    m_timeNow = std::chrono::high_resolution_clock::now();
} // END DER::DER

DER::~DER()
{
    //dtor
} // END DER::~DER

void DER::GetTime()
{
        m_timePast = m_timeNow;
        m_timeNow = std::chrono::high_resolution_clock::now();
        m_timeElapsed = (m_timeNow - m_timePast);
} // END DER::GET TIME

void DER::ImportPower(){
        DER::GetTime();                                        // update elapsed time
    double energy = m_timeElapsed * m_importControl/3600;          // unit=watt-hours
    // Check to see if the battery can support the full import energy
    // If it cannot then force Import Energy = 0% and Export Energy = 100%
    // Else adjust values accordingly.
    if(m_importWattHours < energy){

        m_importWattHours = 0;
        m_exportWattHours = m_ratedExportEnergy;
    } else {
        m_importWattHours = m_importWattHours - energy;
        m_exportWattHours = m_exportWattHours + energy;
    }

}; // END IMPORT POWER

void DER::ExportPower(){
        DER::GetTime();                                        // update elapsed time
    double energy = m_timeElapsed * m_exportControl/3600;          // unit=watt-hours

    // Check to see if the battery can support the full export energy
    // If it cannot then force Import Energy = 100% and Export Energy = 0%
    // Else adjust values accordingly.
    if(m_exportWattHours < energy){
        m_importWattHours = m_ratedExportEnergy;
        m_exportWattHours = 0;
    } else {
        m_importWattHours = m_importWattHours + energy;
        m_exportWattHours = m_exportWattHours - energy;
    }

}; // END EXPORT POWER

void DER::Regulate(){
        DER::GetTime();                                        // update elapsed time

        // Check for import/export regulation
    if(m_regulate > 0){
                m_exportControl = 0;
```

```
                // adjust import control according to regulation
                if(m_importControl < m_ratedImportPower * (-m_regulate)){
                        m_importControl = m_importWatt + m_importChar * m_timeElapsed;
                } else {
                        m_importControl = m_importWatt - m_importChar * m_timeElapsed;
                }
    } else {
                m_importControl = 0;
                // adjust export control according to regulation
                if(m_exportControl < m_ratedExportPower * m_regulate){
                        m_exportControl = m_exportWatt + m_exportChar * m_timeElapsed;
                } else {
                        m_exportControl = m_exportWatt - m_exportChar * m_timeElapsed;
                }
    }
}; // END EXPORT POWER

void DER::Losses(){
    if (m_importWattHours > 0){
        m_exportWattHours = m_importWattHours - m_idleChar * m_timeElapsed;
    } else {
        m_exportWattHours = 0;
    }
}; // END LOSSES
```

A.3.3 bess.h

```
#ifndef BESS_H
#define BESS_H
#include <chrono>        // high_resolution_clock

class DER
{
    public:
        DER();
        virtual ~DER();
        void GetTime();
        void ImportPower();
        void ExportPower();
                void Regulate();
        void Losses();

    protected:
        std::chrono::time_point<std::chrono::high_resolution_clock> m_timePast;
        std::chrono::time_point<std::chrono::high_resolution_clock> m_timeNow;
        std::chrono::duration<double> m_timeElapsed;                // unit=seconds

        double m_importControl, m_exportControl;                   // unit=watts
        double m_importWatts, m_exportWatts;
        double m_importWattHours,
          ↪   m_exportWattHours;                                   // unit=watt-hours
                double m_idleChar, m_importChar,
                  ↪   m_exportChar;                                  // unit=watts
                  ↪   per second
                double m_regulate;

                double m_ratedImportPower = 8000;
                double m_ratedExportPower = 8000;
                double m_ratedImportEnergy = 24000;
                double m_ratedExportEnergy = 21500;

    private:
};

#endif // BESS_H
```

A.3.4 ewh.cpp

```
#include "ewh.h"

DER::DER() :
    m_importControl(0),
    m_exportControl(0),
    m_importWatts(0),
    m_exportWatts(0),
    m_importWattHours(1000),
    m_idleChar(1),
    m_importChar(1000),
    m_exportChar(1000)
{
    //ctor
    DER::Random();
    m_timeNow = std::chrono::high_resolution_clock::now();
```

```
} // END DER::DER

DER::~DER()
{
    //dtor
} // END DER::~DER

void DER::GetTime()
{
        m_timePast = m_timeNow;
        m_timeNow = std::chrono::high_resolution_clock::now();
        m_timeElapsed = (m_timeNow - m_timePast);
} // END DER::GET TIME

void DER::ImportPower(){
    double energy = m_timeElapsed * m_importControl/3600;         // unit=watt-hours
    // Check to see if the battery can support the full import energy
    // If it cannot then force Import Energy = 0% and Export Energy = 100%
    // Else adjust values accordingly.
    if(m_importWattHours < energy){

        m_importWattHours = 0;
        m_exportWattHours = m_ratedExportEnergy;
    } else {
        m_importWattHours = m_importWattHours - energy;
        m_exportWattHours = m_exportWattHours + energy;
    }

}; // END IMPORT POWER

void DER::ExportPower(){
    double energy = m_timeElapsed * m_exportControl/3600;         // unit=watt-hours

    // Check to see if the battery can support the full export energy
    // If it cannot then force Import Energy = 100% and Export Energy = 0%
    // Else adjust values accordingly.
    if(m_exportWattHours < energy){
        m_importWattHours = m_ratedExportEnergy;
        m_exportWattHours = 0;
    } else {
        m_importWattHours = m_importWattHours + energy;
        m_exportWattHours = m_exportWattHours - energy;
    }

}; // END EXPORT POWER

void DER::Regulate(){
        DER::GetTime();                                          // update elapsed time
        // Check for import/export regulation
    if(m_regulate > 0){
                m_exportControl = 0;

                // adjust import control according to regulation
                if(m_importControl < m_ratedImportPower * (-m_regulate)){
                        m_importControl = m_importWatt + m_importChar * m_timeElapsed;
                } else {
                        m_importControl = m_importWatt - m_importChar * m_timeElapsed;
                }
    } else {
                m_importControl = 0;

                // adjust export control according to regulation
                if(m_exportControl < m_ratedExportPower * m_regulate){
                        m_exportControl = m_exportWatt + m_exportChar * m_timeElapsed;
                } else {
                        m_exportControl = m_exportWatt - m_exportChar * m_timeElapsed;
                }
    }
}; // END EXPORT POWER

void DER::Losses(){
    if (m_importWattHours > 0){
        m_exportWattHours = m_importWattHours - m_idleChar * m_timeElapsed;
    } else {
        m_exportWattHours = 0;
    }
}; // END LOSSES
```

A.3.5 ewh.h

```
#ifndef EWH_H
#define EWH_H
#include <chrono>       // high_resolution_clock

class DER
{
    public:
        DER();
        virtual ~DER();
        void GetTime();
        void ImportPower();
        void ExportPower();
```

```
            void Regulate();
        void Losses();

    protected:
        std::chrono::time_point<std::chrono::high_resolution_clock> m_timePast;
        std::chrono::time_point<std::chrono::high_resolution_clock> m_timeNow;
        std::chrono::duration<double> m_timeElapsed;                  // unit=seconds

        double m_importControl, m_exportControl;                      // unit=watts
        double m_importWatts, m_exportWatts;
        double m_importWattHours,
          ↪   m_exportWattHours;                                       // unit=watt-hours
                double m_idleChar, m_importChar,
                  ↪   m_exportChar;                                       // unit=watts
                  ↪   per second
                double m_regulate;

                double m_ratedImportPower = 1500;
                double m_ratedImportEnergy =
                  ↪   1000;                                                              //
                  ↪   TODO: this should be updated by draw profile hourly

    private:
};

#endif // EWH_H
```

A.3.6 Makefile

```
# Copyright AllSeen Alliance. All rights reserved.
#
#    Permission to use, copy, modify, and/or distribute this software for any
#    purpose with or without fee is hereby granted, provided that the above
#    copyright notice and this permission notice appear in all copies.
#
#    THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
#    WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
#    MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
#    ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
#
#    WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
#    ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
#    OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

# Modified by V-SQUARED, June 2016

CPPOPTS :=

ALLJOYN_DIST := $(AJ_ROOT)/build/$(OS)/$(CPU)/debug/dist/cpp

OBJ_DIR := obj/Debug

BIN_DIR := bin/Debug

ALLJOYN_CPPLIB := $(ALLJOYN_DIST)/lib
COMMON_INC := $(AJ_ROOT)/common/inc

CXXFLAGS = -Wall -pipe -std=c++11 -fno-rtti -fno-exceptions -Wno-long-long -Wno-deprecated
  ↪   -g -DQCC_OS_LINUX -DQCC_OS_GROUP_POSIX -DQCC_DBG -DROUTER -O0 -Wno-write-strings

LIBS = -lstdc++ -lpthread -lalljoyn -lajrouter -lrt -lm

default: DMS

DMS:        dms.o der.o
        mkdir -p $(BIN_DIR)
        $(CXX) -o $(BIN_DIR)/DMS $(OBJ_DIR)/dms.o $(OBJ_DIR)/der.o -L$(ALLJOYN_CPPLIB)
  ↪   $(LIBS)

dms.o:        dms.cpp der.h $(ALLJOYN_LIB)
        mkdir -p $(OBJ_DIR)
        $(CXX) -c $(CXXFLAGS) -I$(ALLJOYN_DIST)/inc -o $(OBJ_DIR)/$@ dms.cpp

der.o:        der.cpp der.h
        mkdir -p $(OBJ_DIR)
        $(CXX) -c $(CXXFLAGS) -o $(OBJ_DIR)/$@ der.cpp

all_clean: clean
        rmdir $(OBJ_DIR)
        rmdir $(BIN_DIR)
        rmdir bin
        rmdir obj
        rm *.plt

clean:
        rm -f $(BIN_DIR)/DMS
        rm -f $(OBJ_DIR)/dms.o
        rm -f $(OBJ_DIR)/der.o
```

A.4 BASH Scripts

A.4.1 Appsetup

```
export CPU=arm
export OS=linux
export VARIANT=debug
export ALLJOYN_ROOT=$HOME/src/alljoyn
export AJ_ROOT=$ALLJOYN_ROOT
export LD_LIBRARY_PATH=$AJ_ROOT/build/linux/$CPU/$VARIANT/dist/cpp/lib:$LD_LIBRARY_PATH
```

A.4.2 Run

```
#!/bin/bash
export CPU=arm
export OS=linux
export VARIANT=debug
export ALLJOYN_ROOT=$HOME/src/alljoyn
export AJ_ROOT=$ALLJOYN_ROOT
export LD_LIBRARY_PATH=$AJ_ROOT/build/linux/$CPU/$VARIANT/dist/cpp/lib:$LD_LIBRARY_PATH
./../cpp/bin/Debug/DMS 1
```

A.4.3 Run Multiple

```
#!/bin/bash
source appsetup
#compile app
make -C ../cpp
#run lots of apps
num="$1"
if [ "$num" = "" ];
then
        echo You must enter the number of clients
else
        n=0
        while [ $n -lt $num ];
        do
                tmux new-window -d ./run.sh
                sleep 5
                let n+=1
        done
fi
```

A.4.4 System Log

```
#!/bin/bash
file="/home/tylor/dev/EMS/tools/test2/PClog$(date +%R).txt"
file2="/home/tylor/dev/EMS/tools/test2/NETlog$(date +%R).txt"
tshark -a duration:30 -i br0 -w $file2 &
echo "MEM, CPU" > $file
n=0
while [ $n -lt 30 ];
do
        MEM=$(free -m | awk 'NR==2{printf "%.2f%%, ", $3*100/$2}')
        CPU=$(top -bn1 | grep load | awk '{printf "%.2f%%", $(NF-2)}')
        echo "$MEM$CPU" >> $file
        sleep 1
        let n+=1
done
```