A Communication Testbed for Testing Power Electronic Agent Systems

Benjamin Dean[1], Michael Starke[2], Mitchell Smith[2], Madhu Chinthavli[2], Leon Tolbert[1]
University of Tennessee, Knoxville, TN, USA
Oak Ridge National Laboratory, Oak Ridge, TN 37831
{bdean7, tolbert}@utk.edu, {starkemr, smithmt, chinthavalim}@ornl.gov

***Abstract*—Power electronic systems (PES) incorporate complex intra-system communication, which are of vital importance for the successful operation of these systems. This paper proposes and outlines a communication testbed that will help in the development and testing of the communications between the components of PES. It allows for the comparison and evaluation of different communication methods, such as MQTT, Modbus, or User Datagram Protocol (UDP), and for the characterization of how these communication protocols perform.**

***Index Terms*— power electronic systems, communications, UDP, MQTT, Modbus**

I. INTRODUCTION

The electric power grid is transitioning to an electric network consisting of power electronic systems (PES), communication systems, and intelligent devices. By 2030, the prediction is that 80% of the power generated will flow through PES [1]. As PES continue to see rapid adoption, the complex nature of system interactions and couplings will continue to grow. New systems for interconnecting energy sources, energy storage, and loads are in rapid development and deployment. In adopting these technologies effectively, communication and control approaches will be paramount.

When developing communication for PES, both the communication protocol and schema affect the operation of the system. The schema defines how the data is interpreted and translated in the communication protocol. Examples of communication protocols include Message Queuing Telemetry Transport (MQTT), Modbus, and UDP. When developing PES, the latency, error rate, and message rate of the communication protocol and schema are important for the development of the PES control. Therefore, quantifying these characteristics is important before hardware testing is done. In Fig. 1, a design tree is proposed showing the development from concept to system prototype by including an additional layer of development and testing of the communication network before Hardware-in-the-loop (HIL) integration, therefore allowing for the communication characteristics to be incorporated into the system control.

In this work, a testbed for communication networks within a modular power electronic system of distributed controllers is discussed. This testbed provides an opportunity to develop and implement new communication and control strategies while also evaluating the security, latency, and reliability of the solution.

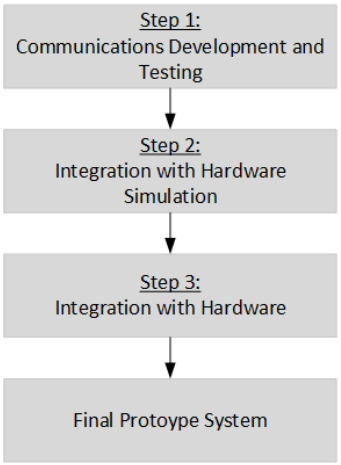

Fig. 1. Process flow in the development of power electronic system.

The testbed is used to evaluate the communication between a computational node and a digital signal processor (DSP). PES utilize multiple computational nodes that communicate with one another and perform various tasks in a system in a decentralized manner. These tasks include interfacing with the outside world, interfacing with DSPs that control power converters, and running system optimization. This testbed provides an environment to test and implement the communication schemes of these systems. Latency, message rate, and error rate can quickly be determined, and PES control can be adjusted to fully utilized the communication capability available.

Fig. 2 shows a typical configuration of an agent-based power electronics system. These different sub-systems work independently to accomplish an overall goal, such as economic gain, or grid stability. Agent-based systems provide expandability and versatility in deployment and operation and have been shown to support power electronic system integration [2]-[3]. Each integrated component can be controlled by an agent and provide it with specific intelligence and decision making. Agents in these systems must demonstrate the ability to be reactive, proactive, and social. Agents need to perceive local conditions and react accordingly,

initialize and exhibit goal-directed behavior, and interact with other agents or external entities [4].

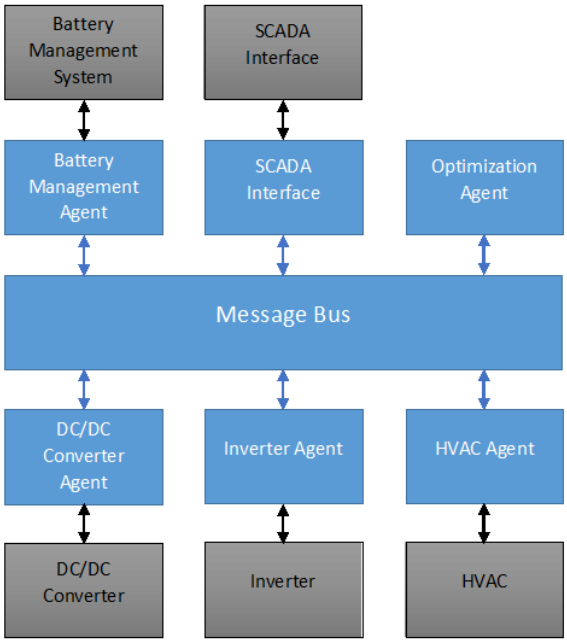

Fig. 2. Basic Agent-based communication implementation of a PES. Some or all these example agents could be incorporated

The blue arrows represent the agent communication. Message-bus communication can take place over protocols such as MQTT, VOLTTRON, or OpenFMB. The black arrows represent the agent to component communication. This can also happen over a variety of protocols, such as Modbus TCP, Modbus RTU, or UDP.

II. Agent Based Systems

Various agent-based systems with different communication implementations have been presented in literature. These systems were reviewed to help determine the communication protocol and configuration this testbed should be capable of testing. In [2]-[3], agent systems were developed to link power electronic converters (PEC) and resources into a single system available for dispatch by a central coordinator. The agent systems turned single requests into actionable functions between different systems and lead to fully coordinated systems. Even negotiation techniques have been investigated with this type of schema [5]. This can be further complicated with a hierarchy of system needs to support the electric grid as presented in [6]. Here the work proposes interconnection of utility distribution energy management system (DeMS) to power electronic systems through OpenADR and a home energy management system. Hence, the driving features required of the power electronic system will create the necessary communications framework, protocol, and schema for the communication network.

In previous work, a plug-and-play framework solution was developed for conversion stages in a PES. This allows control and data to be transferred from a digital signal processing (DSP) controller to a central controller regardless of the converter type or design [5]. In Fig. 2, the communication between a DSP and a middle layer (computer node or Raspberry Pi) is presented. Communication between the computer node and DSP is achieved using a standardized implementation of UDP. The communication data includes specification, available control modes, and converter ratings to support auto-configurability [5].

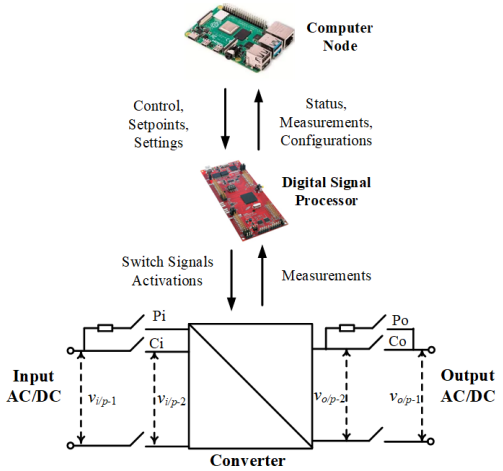

Fig. 3. Example integration with real hardware [8]

Within the computer node lies an agent framework, a locally hosted broker (message bus) on a single computational node is used by the agents for MQTT communication. The framework utilizes 4 unique agents: 1) the Source/Load agent that communicates and interacts with interconnected sources or loads, 2) the Converter agent which communicate to the converter DSP through UDP, 3) the Intelligence agent which determines the commands that will be dispatched to the converter and source/load and 4) the Interface agent which is used to link the computational node to a central controller. In this architecture, the computer node also communicates via MQTT to the central controller but can support other protocols.

III. Communications Testbed

A. Overview

For evaluating different functionalities and communication protocols, a communication testbed must be able to emulate the schema and protocol of the different PES components and evaluate the performance of the communications. This provides a rapid means to test various case scenarios, debug setup challenges with communication configurations, and establish communication baselines for comparison. For effective results, the network configuration must be the same as the PES network implementation. Therefore, the testbed utilizes two separate networks for the PES communications emulation, and one for the central computer to interact with the Raspberry Pi's. The code deployed is in the form of agents in a Raspberry Pi network, with different protocols able to be used as the DSP to computer node communications and MQTT for the communications between computer nodes and central computer. Custom schemas for the DSP to computer node and MQTT communications can be defined, deployed, and tested using the developed graphical user interface (GUI) .

As presented, three sets of two Raspberry Pi 3B models are used to test the computational node to DSP communication, with one set of Raspberry Pi's representing interconnected DSPs and the other the computer nodes. This communication link is an isolated ethernet layer to provide direct link between the computer node and DSP simulator. The computational nodes (Raspberry Pi's A/B/C) share an ethernet switch for intercommunication with MQTT. An Arduino Uno is interfaced with the general-purpose input/output (GPIO) pins of the Raspberry Pi's. This allows for timing experiments, such as latency testing, without requiring Raspberry Pi clock synchronization. The following is a list of the hardware used in the testbed, pictured in Fig. 4:

- Raspberry Pi 3B (3) – Computer nodes
- Raspberry Pi 3B (3) – DSP Simulator nodes
- Arduino Uno
- Network Switch (2)
- MacBook Pro running Ubuntu 16.04 (Code Deployment)
- MacBook Pro running Ubuntu 16.04 (Central Computer)

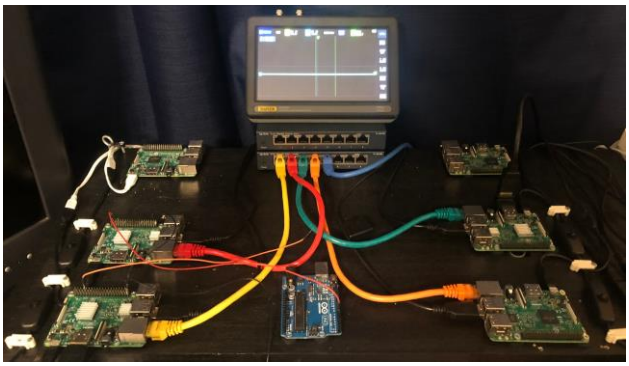

Fig. 4. Photo of the testbed setup during latency testing

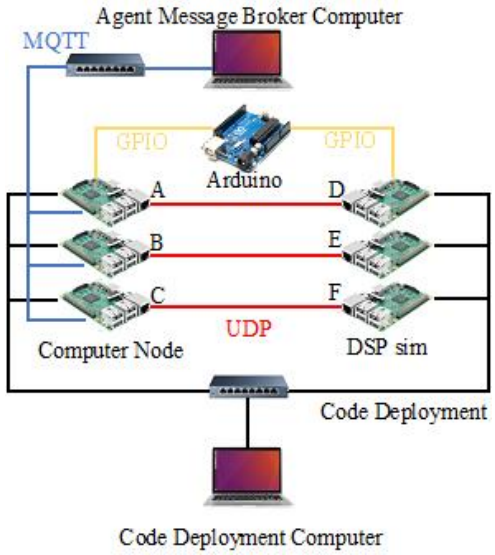

Fig. 5. Diagram of the communications testbed with Raspberry Pi nodes, Arduino Uno, central computer, and code deployment computer

Since the Raspberry Pi's utilize more than a single ethernet connection, USB to ethernet adapters were employed to support additional communication layers as shown in Fig. 6. The Raspberry Pi automatically adopts an *eth0*, *eth1*, and *eth2* naming convention for communication ports allowing for rapid adoption and evaluation of the communication port. In this case, *eth0* is used for the UDP communications, *eth1* for the communication network to support automatic deployment of code across the network for testing, and *eth2* for MQTT communications between computer node and central computer.

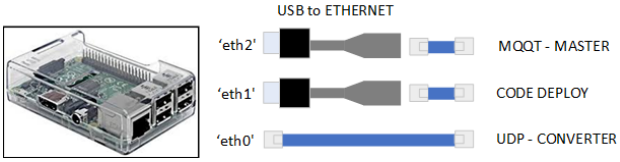

Fig. 6. Diagram of the different communication connections to support

B. *Communication Networks*

The setup of the testbed uses a Code Deployment Computer (CDC) running Ubuntu 16.04 that is connected to all the Raspberry Pi's. This network is used for detecting and auto-deploying code to the Raspberry Pi's. Each node on this network has a static IP address defined with the form 192.168.0.X (*eth1*). The netmask is 255.255.255.0, which limits the communication network addresses 192.168.0.0 – 192.168.0.255. The communication for *eth0* is always set to 192.168.53.X for the computer nodes and the DSP simulators (as the communication link is isolated). The MQTT communication network is preconfigured as well with 192.168.99.X and netmask 255.255.255.0.

The code deploy network allows the following tasks to be facilitated without interfering with the communication network.

- Detecting connected Raspberry Pi's through pinging
- Transferring scripts to Raspberry Pi's through a network file system (NFS)
- Executing Python scripts on the Raspberry Pi's through secure shell (SSH) commands
- Remote access setup for troubleshooting through Virtual Network Computing (VNC)

By implementing the communication networks with the same configuration as physical PES to test the system behavior, the communication behavior can be determined in different real-world situations, including cyber-attacks. MQTT supports secure socket layer (SSL) encryption for secure communication, and the testbed can be configured to incorporate this into its implementation. This allows cyber-attacks to be emulated, and the resiliency of the communications to these attacks can be determined.

C. *Evaluation techniques for establishing quaility of communications*

Two metrics were established as important for communication: 1) maximum number of communication

messages without data loss and 2) latency associated with the communicating and processing of messages.

1) Maximum Speed of Communications without Data Loss

The error rate of messages with the UDP communication socket needed to be determine, as UDP is a "connectionless" communication protocol that does not ensure data is received. Other protocols, such as Modbus or MQTT, have incorporated data verification, and therefore do not need their error rate tested. However, this verification limits their maximum messages per second.

To determine the maximum UDP messages per second, a python script was developed on the testbed to send a specified number of UDP messages per second. The messages per second started at 300, and incrementally increased to a maximum of approximately 9000 messages. The sent and received data was compared to determine how many messages were properly received.

$$error\ rate = \frac{messages\ sent - messages\ received}{messages\ sent} \quad (1)$$

2) Anticipated Speed of Communications

The speed of the data transfer from point to point is important, as it can be a limiting factor in PES control. Two factors limit the speed in which data is sent and processed, the network latency and the computational speed of the Raspberry Pi 3B. The data flow of a UDP message is shown in Fig. 8. The data flow is as follows:

- Push button: The action taken by user or agent to initiate sending a message.
- Encode Message (t_{encode}): The message data is encoded into 32 bits that contains the category, subcategory, and value. (i.e. Control/Output Voltage/120)
- Send/Receive Message ($t_{latency}$): The 32-bit data is processed, sent, and received over the network using the "socket" library in Python 3.
- Decode Message (t_{decode}): The 32 bits are decoded into the category, subcategory, and data value

The total message time can be computed as:

$$t_{message} = t_{encode} + t_{latency} + t_{decode} \quad (2)$$

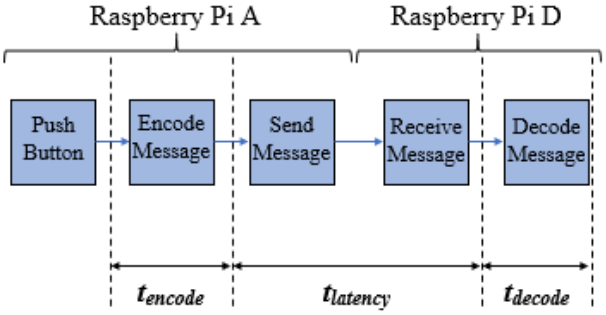

Fig. 7. The data flow of messages with the custom UDP communication scheme

To measure the times accurately, GPIO pins of the Raspberry Pi's were used. Based on the value being measured, a GPIO pin switched between "HIGH" and "LOW." The pulses were measured by an Arduino Uno microcontroller. This allowed the Raspberry Pi's to not be synchronized which latency testing. The accuracy of the Arduino was verified using a Yeapook ADS1013D oscilloscope, as demonstrated in Fig. 5, which shows the testbed during latency testing. Based on testing, the average time for a Raspberry Pi 3B to activate a GPIO pin was <2µs, and therefore was disregarded due to its small value.

IV. Results

This communication testbed was used to determine the maximum speed and potential latency associated with using the Raspberry Pi 3B and developed UDP schema. The UDP IPv4 implementation requires 32 bits for the sending and receiving address, 32 bits for the length and checksum data, and 32 bits for the PES data. Based on the Raspberry Pi's maximum documented ethernet networking speed of 10 mbps, this gives a theoretical limit of 104167 messages/second [7]. The measured messages per second verses the error rate is shown in Fig. 9.

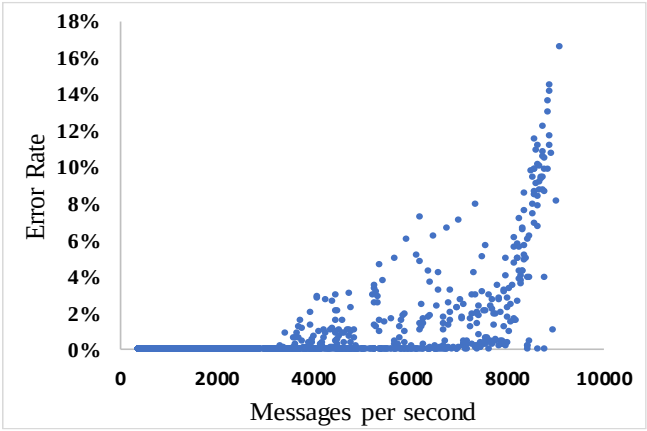

Fig. 8. Messages sent per second verses error rate

This test showed that the Raspberry Pi can send approximately 9000 messages per second, well short of the network capability. However, when messages exceeded 3325 per second data loss was observed. Fig. 10 shows the overall message time $t_{message}$ captured while sending 3000 messages with no loss. Table I shows the breakout of the communication components that represent the message. The average overall message time was measured to be 506 µs. The maximum message rate is limited by the time required to encode the UDP message, as:

$$\frac{1\ message}{t_{encode}} = \frac{1\ message}{112\ \mu s} = 8928\ messages/second \quad (3)$$

and network latency as:

$$\frac{1\ message}{t_{latency}} = \frac{1\ message}{292\ \mu s} = 3424\ messages/second \quad (4)$$

In this setup, the network latency plays the largest role in limiting the maximum message rate (which is limited by the ethernet port on the Raspberry Pi). However, clearly the computational capabilities of the Raspberry could also impose additional limits based on other computational needs for the agent system.

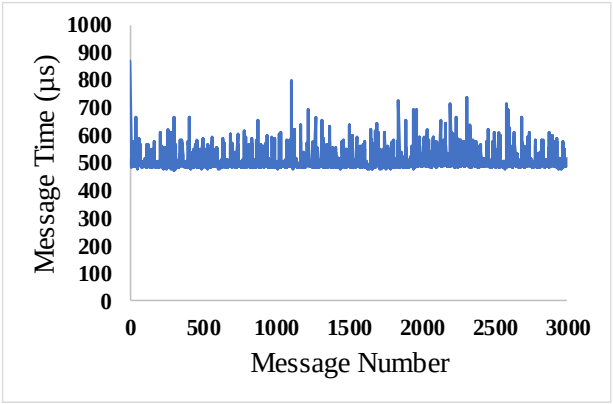

Fig. 9. UDP message time $t_{message}$ over 3000 messages

TABLE I. AVERAGE TIMES

t_{encode}	112 µs
$t_{latency}$	292 µs
t_{decode}	102 µs
$\boldsymbol{t_{message}}$	**506 µs**

I. CONCLUSION

This paper covers the need, requirements, and development of a testbed to help aid the deployment of an agent system supporting power electronic systems. The testbed streamlines the process of communications evaluation and testing with auto deployment functionality.

Furthermore, this paper demonstrates use of the testbed to examine a new UDP communication schema implemented with a Raspberry Pi 3B. The maximum error-free message rate and message time of this communication scheme was found with the testbed and quantified.

Future work with this testbed will include validating the full communication messaging, from master controller to end node to establish baselines for the ability to respond to controls quickly. This work will encompass MQTT, UDP, and Modbus. Future work also includes testing with different computer platforms, such as the Raspberry Pi 4, which has faster network speed and more computational power. By using newer hardware, the data transfer limits on low-cost single-board computers for future PES can be validated and implemented. Last, cybersecurity protocols, such as SSL for UDP, will be explored.

II. ACKNOWLEDGMENTS

This work was funded by the U.S. Department of Energy, Office of Electricity, Energy Storage Program under contract number DE-AC05-00OR22725.

REFERENCES

[1] L. M. Tolbert et al., “Power Electronics for Distributed Energy Systems and Transmission and Distribution Applications,” Ornl/Tm-2005/230, no. December, pp. 65–91, 2005

[2] M. Starke et al., “A Multi-Agent System Concept for Rapid Energy Storage Development,” 2019 IEEE Power Energy Soc. Innov. Smart Grid Technol. Conf. ISGT 2019, pp. 1–5, 2019

[3] M. Starke et al., “Residential (Secondary-Use) Energy Storage System with Modular Software and Hardware Power Electronic Interfaces,” 2019 IEEE Energy Convers. Congr. Expo. ECCE 2019, pp. 2445–2451, 2019

[4] N. A. Z. Musa, M. Z. M. Yusoff, R. Ismail, and Y. Yusoff, “Issues and challenges of forensics analysis of agents’ behavior in multi-Agent systems: A critical review,” 2015 Int. Symp. Agents, Multi-Agent Syst. Robot. ISAMSR 2015, pp. 122–125

[5] M. T. Smith, M. R. Starke, M. Chinthavali, and L. M. Tolbert, “Architecture for Utility-Scale Multi-Chemistry Battery Energy Storage,” 2019 IEEE Energy Convers. Congr. Expo. ECCE 2019, pp. 5386–5392, 2019

[6] M. Starke et al., “Networked Control and Optimization for Widescale Integration of Power Electronic Devices in Residential Homes,” 2019 IEEE Energy Convers. Congr. Expo. ECCE 2019, pp. 3496–3501, 2019

[7] J. Geerling, “Getting Gigabit Networking on a Raspberry Pi 2, 3 and B+.” https://www.jeffgeerling.com/blogs/jeff-geerling/getting-gigabit-networking (accessed Sep. 25, 2020).

[8] M. Starke et al., “Agent-based framework for supporting behind the meter transactive power electronic systems,” 2020 IEEE Power Energy Soc. Innov. Smart Grid Technol. Conf. ISGT 2020, 2020