Pacific Northwest NATIONAL LABORATORY

Attivo NETWORKS®

PNNL-30387

Model Driven Deception for Defense of Operational Technology Environments

September 2020

Thomas W. Edgar
William Hofer
Marc Feghali

U.S. DEPARTMENT OF ENERGY

Prepared for the U.S. Department of Energy
under Contract DE-AC05-76RL01830

DISCLAIMER

PACIFIC NORTHWEST NATIONAL LABORATORY
operated by
BATTELLE
for the
UNITED STATES DEPARTMENT OF ENERGY
under Contract DE-AC05-76RL01830

Printed in the United States of America

Available to DOE and DOE contractors from the
Office of Scientific and Technical Information,
P.O. Box 62, Oak Ridge, TN 37831-0062;
ph: (865) 576-8401
fax: (865) 576-5728
email: reports@adonis.osti.gov

Available to the public from the National Technical Information Service
5301 Shawnee Rd., Alexandria, VA 22312
ph: (800) 553-NTIS (6847)
email: orders@ntis.gov <https://www.ntis.gov/about>
Online ordering: http://www.ntis.gov

PNNL-30387

Model Driven Deception for Defense of Operational Technology Environments

September 2020

Thomas W. Edgar
William Hofer
Marc Feghali

Prepared for
the U.S. Department of Energy
under Contract DE-AC05-76RL01830

Pacific Northwest National Laboratory
Richland, Washington 99354

Summary

There is an ever-increasing number of cyber-attacks targeted at cyber-physical systems vital to the operation of our critical infrastructure. Everything from disruption (Lee, Assante, and Conway 2016), destruction (Cyber.nj.gov 2017), data loss (Greenberg 2018), or general rampant internet threats (Honeywell 2018) have become a risk to cyber-physical systems that were once thought isolated and secure from cyber threats. As with the advent and proliferation of Internet in the 90s, deploying cyber defenses is critical to robust and resilient operation.

Deception defense is a possible solution. Deception defense is a security technology that is low impact to the operations while providing benefits for both incumbering active threats and boosting defender awareness. Deception defense fits well within the limitations of OT environments. Integrating decoys into existing installments is low impact and relatively easy. Deceptions are installed and configured around existing system components limiting the risk of impacting availability and operational performance. In addition, decoys are designed to draw away the attention of attackers from existing systems with known weaknesses to reduce the likelihood of impact on operations due to threat infiltration. Decoys give valuable time to defenders to mitigate and respond to threats actively attempting to cause impact on their most critical systems.

However, traditional deception needs enhancements to appear realistic and be effective within OT environments. In this whitepaper research and development into high fidelity deception of field devices using model driven simulations is presented. The features presented will be discussed in the context of an electrical distribution substation and as they have been implemented within the Attivo Networks BOTsink platform.

A high-fidelity OT decoy has three main attributes which include services, variables, and behavior. Each of these attributes defines the characteristics of the decoy acting as a device. Breaking down each decoy into these three attributes allows for an abstraction of complex system architectures. A device in a real environment would speak one or more network protocols, control or monitor some set of variables, and perform actions based on a set of logic. To be effective, a decoy should also do the same. Each of these attributes are user defined to construct decoys that appear as a realistic device in their system.

Due to the strong integration of real-world physics, OT deception platforms must operate differently than traditional IT deceptions. For instance, turning off a valve will be detected downstream by other sensors because the flow will reduce and stop. Additionally, controllers and applications leverage data from sensors to send control commands to each other. A believable deception must be integrated with the system to project the effects of events. An attack will likely attempt to control the physical process in a negative manner. To make the attacker believe they are achieving their objective, it must predict the effects of these actions, to a reasonable degree. Our approach to simulating a model to generate realistic decoy behavior is explored including description of two approaches: a physics model-based approach and a data driven approach. The performance of two machine learning techniques are investigated in their ability to learn a good enough model of the physics of the system.

Acknowledgments

The research described in this whitepaper was funded by the Department of Energy Office (DOE) of Office of Technology Transitions (OTT) technology commercialization fund. The original research that led to this project was part of the Proactive Adaptive Cybersecurity for Control Agile (PACiFiC) Initiative at Pacific Northwest National Laboratory (PNNL). It was conducted under the Laboratory Directed Research and Development Program at PNNL, a multi-program national laboratory operated by Battelle for the U.S. Department of Energy.

Contents

Figures

Tables

1.0 Introduction

There is an ever-increasing number of cyber-attacks targeted at cyber-physical systems vital to the operation of our critical infrastructure. Everything from disruption (Lee, Assante, and Conway 2016), destruction (Cyber.nj.gov 2017), data loss (Greenberg 2018), or general rampant internet threats (Honeywell 2018) have become a risk to cyber-physical systems that were once thought isolated and secure from cyber threats. As with the advent and proliferation of Internet in the 90s, deploying cyber defenses is critical to robust and resilient operation.

Common IT cyber controls are not all equally effective within cyber-physical systems or operational technology (OT), however. The unique aspects of OT systems like unique protocols, limited resources, embedded system to system operation, and extreme high availability requirements limit the ability to apply common IT defensive techniques like continuous patching and ubiquitous and current cryptography. As such, new cutting-edge defenses are needed that easily integrate and provide high defensive value to combat the increasing cyber risks for OT systems.

Deception defense is one possible solution to this situation. Deception defense is a security technology that is low impact to the operations while providing benefits for both incumbering active threats and boosting defender awareness. With deception defense, realistic decoys of systems and data that appear to be operational are deployed within an organization. These decoys are often designed with data or weaknesses to lure attackers to interact with them and waste their time and resources exploiting them and using their data to further exploit other decoys. These decoys act as canaries that alert defenders to the presence of threats within their networks. As the decoys are not truly operational, internal services and users should not be interacting with them resulting in high accuracy threat detection with very low false positive rates as compared to other situational awareness tools.

Deception defense presents a solution that fits well with the limitations of OT environments. Integrating decoys into existing installments is low impact and relatively easy. Deceptions are installed and configured around existing system components limiting the risk of impacting availability and operational performance. In addition, decoys are designed to draw away the attention of attackers from existing systems with known weaknesses to reduce the likelihoodof impact on operations due to threat infiltration. Decoys give valuable time to defenders to mitigate and respond to threats actively attempting to cause impact on their most critical systems.

However, traditional deception needs enhancements to appear realistic and be effective within OT environments. The rest of this whitepaper details the various features and capabilities developed to make high fidelity, model driven deception for defense of OT systems. The features presented will be discussed in the context of an electrical distribution substation and as they have been implemented within the Attivo Networks BOTsink platform. Experimental results of performance of these features is discussed at the end of the whitepaper.

2.0 Deception for OT

Cyber deception has been an academic concept since the late 1980's (Stoll 1989) and was first developed into a security tool by Cohen et. al (2001). Since inception, cyber deception has been commonly used as tools for cyber threat intelligence gathering on the internet. Low to high interaction honeypots have been operated to collect and study the current trend of malware and threat targets and tactics. The honeynet.org (Honeynet Project 2019) project has become a major community organization for organizing and cataloging honeypot contributions. Historically, research to develop and use honeypot technology has been focused on traditional IT environments. With the increasing interest in cyber-physical environments over the last decade, focus on deception research for these environments has also increased.

It is understood that OT and IT systems have different operational and cyber security requirements (Stouffer et. al. 2011). The best practice of rolling updates and patch cycles that are now commonplace in enterprise operations are much more difficult to follow in OT environments due to the high availability requirement and subsequent rigorous and slow testing and validation processes. The high response time requirements to defuse physical safety risks has led to low penetration of strong access controls and cryptography that is ubiquitous in IT. As such, additional cyber security techniques are needed to provide OT defenders the ability to protect themselves from cyber threats. Cyber deception is a solution that works around and within these requirements that provides a tool to counter the efforts of threats and to enhance the detection and response times of defenders.

The initial effort to develop an OT focused honeypot was the SCADA Honeynet Project (Pothamsetty and Franz n.d). This effort leveraged the honeyd low interaction honeypot daemon that provides the ability to create simulated hosts with network services on a system. The SCADA Honeynet project extended honeyd to model a common OT programmable logic controller (PLC). Since then a few more OT specific honeypots have been developed targeting different domains and use cases. The Conpot project (Rist et. al. 2018) is a docker based low interaction honeypot designed to mimic common industrial communication protocols to appear as process control devices. The GasPot project (Willhoit and Hilt 2015) created, from scratch, a special purpose python-based honeypot to emulate a Veeder Root Guardian AST. Both MiniCPS (Antonioli and Tippenhauer 2015) and HoneyPhy (Litchfield 2016) are research into frameworks to enable model backed ICS honeypots. MiniCPS utilizes the miniNET network emulator with an integrated database to enable the definition and execution of physical process algorithms to feed data to honeypots. HoneyPhy defines an architecture where honeypots can query a simulator to respond with realistic data.

With the strong progression of cloud technologies, more recently there has been a resurgence of using deception techniques to provide cyber network defense (Simoes et. al. 2013). Distributed deception platforms provide centralized control of resources to define, deploy, and manage decoys in operational environments for threat detection and defense (Pingree 2015). A key defining difference between distributed deception platforms and honeypots is that they are meant to be integrated into operational systems for defending against active threats and attacks where honeypots are traditionally used for threat intelligence gathering. In the rest of this paper we present our research into turning concepts from ICS honeypots into a distributed deception platform for integration of decoys into and defense of operational technology systems.

However, two key features are still missing from the academic and commercial deception space to enable deception defense in OT systems: high fidelity OT focused decoys and model driven

behavior and response. A large part of OT environments are embedded computer systems with specific profiles of services and features. It is these specific services and features that are often the targets of OT malware and threats.

3.0 High Fidelity Decoys

A cyber-physical system, by definition, integrates cyber components to monitor and control physical processes. The cyber components are, in general, controllers, sensors, communication infrastructure, and the software applications that use the data from the physical process to perform different functions such as optimization (Younis and Moayeri 2017), delegation of human agency (Subbarao et. al. 2013), or safe process management (Washington State University n.d). The physical systems include the mechanisms that interact with the physical world like actuators and instrumentation that are driven by physical processes. These components commonly consist of programmable logic controllers (PLCs), remote terminal units (RTUs), intelligent electronic devices (IEDs), and process specific sensors. Each type of device performs specific functions within a physical process and are all potential targets of cyber-attacks.

The population of OT devices is broad and spans many industries. However, there are some consistent similarities that differentiate them from IT equipment that are important to understand when developing decoys. First, OT devices are generally embedded cyber-physical systems that have specialized hardware and application programming interfaced operating systems. Second, OT devices generally maintain a connection with a supervisory system for sharing status and enabling remote control. The protocols used for these connections are unique to OT environments and potentially to sectors. For instance, the DNP3 protocol is commonly used only in electrical sector and BACnet in building sector. Third, OT devices provide configuration management and provisioning application programming interfaces. The configuration management interfaces can range from controlled services behind common protocols like telnet and File Transfer Protocol (FTP) to proprietary protocol interfaces. Finally, attacks generally target the device and not just the service running on the device. Where malware in IT environments looks for vulnerable network service versions, OT malware looks for types of devices to target that interface with processes of interest. All these features must be considered when developing OT device decoy profiles.

The three main attributes to our OT Decoys include services, variables, and behavior. Each of these attributes defines the characteristics of the decoy acting as a device. Breaking down each decoy into these three attributes allows for an abstraction of complex system architectures. A device in a real environment would speak one or more network protocols, control or monitor some set of variables, and perform actions based on a set of logic. To be effective, a decoy should also do the same. Each of these attributes are user defined to construct decoys that appear as a realistic device in their system.

OT devices speak a multitude of different protocols depending on the domain in which they are performing control operations, a small sample of which includes DNP3, BACnet, and Modbus protocols. For instance, a decoy in a deception based on building automation and control would most likely communicate via BACnet in a way that relates to the physical process. Devices also support traditional protocols, like HTTP, SSH, Telnet, and FTP, for configuration management. A set of protocols should be associated with each decoy to provide interfaces for threat interaction.

Decoys should expose through protocols one or more variables that correspond to the physical process they are emulating. Decoy variables can include things like temperature, voltage, or flow rate. Any number of variables can be assigned to a decoy and the variables can be newly defined from extrapolations of other existing variables learned from the physical system. A

decoy variable might be an input or an output. For instance, if we want a decoy that monitors pressure and triggers a control valve, we could extrapolate the pressure value by using two input variables from the values for temperature and flow rate from real sensors. A variable that is an output of one decoy device can be read by another device using the network protocol associated with those devices. This interaction generates traffic on the network and adds to the realism of the deception.

The final attribute for an OT Decoy is a set of logic that directs the behavior of the device. A simple example of such logic would be the decision to turn on or off a fan in an HVAC system to allow cool air to flow into a room. A decoy device could act as the controller that monitors the room's air temperature and triggers a fan to rotate if the temperature reaches a certain threshold. The logic of a device can be as simple as an if-then statement. For example, "if temp < X then turn off fan". The logic of decoy devices effects the operation of the simulation which in turn drives the values of the decoy.

Configuring a supported set of these various attributes allows for a flexible mechanism to create decoys that represent a broad range of real devices. Profiles can be created to define those attributes that should be configured for high fidelity decoy representative of real equipment. A device profile is a collection of descriptive metadata and a set of network services which exposes that metadata along with operational system data. In simple terms, a device profile includes the definition of collection of items like model name, firmware ID, and configuration and log files. These metadata sources are then mapped to whatever network services that device supports. For instance, configuration and log files may be exposed through FTP or an HTTP service. There are also behavioral characteristics that are recorded in the metadata that inform how the network services should respond under various conditions. For instance, what commands are allowed in a terminal service telnet session, or what functions are supported in a supervisory control and data acquisition (SCADA) protocol.

3.1 Decoy Services

Services are the applications that provide interfaces and interaction on a network. Services are the method by which an OT device exposes the data it collects, controls to manipulate the process, and configurations on how it should behave. Every OT device supports a collection of services necessary to operate and perform its required functions. In general, there are three general categories that all OT device services fit within: configuration management, process control, and human machine interfacing (HMI). Configuration management services provide an interface to allow engineers to maintain and operate an OT device. Configuration management services practically provide some API to read and write data to a configuration file or database. The configuration file is then used to define what other features and services are utilized during operations.

3.1.1 Configuration Management Examples

In our examples, we have developed a few decoy services to represent equipment from multiple vendors including Schweitzer Engineering Laboratories (SEL), General Electric (GE), and ABB.

3.1.1.1 Telnet and SSH

Terminal services are often available features among OT devices. They are a way of interacting with a device remotely and the functionality available to users can vary wildly depending on the privilege level they log on with. Two different ways of accessing a terminal on a device are

through telnet and SSH. Depending on how the device is configured, telnet and SSH may lead a user to different or the same terminal session, complete with their own sets of login requirements and commands available to different user accounts. Legitimate operators of OT devices might use SSH or telnet as a means of connecting to a terminal on a device to manage settings and configuration. It is not out of the question that an adversary would seek access to a device's terminals, either through telnet or SSH, for nefarious reasons.

Of the devices that have a telnet or SSH service as part of their feature set, the exact behavior one would witness while interacting with it through a terminal has been mimicked as an additional layer within a decoy. A clone of the telnet or SSH service can help to fool an adversary that they have successfully gained access to a vulnerable part of an OT device. Initial efforts to copy telnet functionality consisted of manually going through the commands available to a device while logged on as a specific user account and taking note of the output of those commands. This approach was not quite scalable, as each device may contain dozens of uniquely behaving commands with large outputs. A transition to automation was made to accommodate the large quantity of devices that have telnet services in need of being copied, compounded by the variety in behavior between devices.

The emulated behavior of each device's telnet service is provided through a python script. The script refers to a json file comprised of the list of user accounts on the device, which commands each user has available, and the output of each command when executed at that specific privilege level. The users are mapped to default user accounts configured in the device. The contents of the json file for each device can be entered manually or by using another python script that helps quicken the process of learning telnet behavior. The helper script has two styles available for data collection. The first allows a user to manually connect to a telnet service and learn all behavior as the user interacts with the service as anyone would normally, that is by logging into accounts and sending all commands from those privilege levels. As the user sends commands, the script will save the responses in a json file under the corresponding user account. The second style is fully automated, given account credentials to log in as, along with either a "help" command to initially list the commands the script will go through or a custom list of commands given to the script in advance. Either method will result in a json file that contains the information needed to fully mimic the telnet service for that device.

3.1.1.2 Web server

OT devices generally leverage web servers for two functions: human-machine interface and configuration management. When providing human-machine interface functions, the webserver provides current status, such as register values and protection states, and possibly the ability to manually control, like tripping a relay breaker. The configuration management function allows monitoring and changing the current configuration details, for example communication port details (protocols, addresses, etc.), register to protocol mappings, protection and automation logic, and login credentials. Both functions provide valuable data access from a cyberthreat perspective and therefore must be replicated for high-fidelity decoys.

From an information gathering perspective, the web servers provide a wealth of data about a device. Cyberthreats can use web server data for device targeting and system configuration. It is necessary to both clone the metadata about the device as well as fake the collected operational state of the system (model simulation discussed in future documents). Per these requirements it is necessary to scrape html pages from real devices to provide high-fidelity decoy profiles.

Some of the web pages require authentication and dynamic content. For the authentication it is important to capture what credentials are used to determine the tactics of the threat (stolen credentials, brute force guessing, default attempts, etc.). As such it is important to leave intact, in a limited operational form, the authentication services to collect the attacker actions for analysis. However, it is also important to let the threat progress to keep attackers interested in decoys and to see what additional tactics might be employed. It is therefore designed such that web server authentication mechanisms will allow access after a configured number of failed attempts. Once the threat has logged in it is necessary to fill in the dynamic data expected. Data from real configured devices were scraped to give a realistic starting point. Dynamically updated values need to be faked through server-side scripts or with a simulated physics model.

3.1.1.3 FTP and variants

FTP is a file transfer protocol that allows for uploading and downloading files from OT devices. There are multiple variants of FTP that have been developed overtime and most of them are present across the spectrum of OT devices. These variants include the trivial file transfer protocol (TFTP) which is a simplified version of FTP with less functionality. File transport protocol security (FTPS) and secure file transport protocol (SFTP) both add security to FTP. FTPS is the same as FTP but utilizes an SSL/TLS secure connection mechanism. SFTP on the other hand builds FTP like services on top of the Secure Shell (SSH) protocol.

As mentioned previously, one of the ways in which devices are configured is through FTP or other similar file transfer services (TFTP, SFTP, etc.). The files pushed and pulled to and from the device specify configurations related to communication port details (protocols, addresses, etc.), register to protocol mappings, protection and automation logic, and login credentials. Similarly, to the perspective on web content, this information is valuable from a cyberthreat perspective. By uploading and overwriting a single configuration file an adversary could potentially force open or close a breaker.

Emulating a device with such behavior requires content provided by the decoy FTP service to match the content of a real device. If an adversary lists the file and folder contents and expects to see a certain structure and content, and the decoy is inaccurate, the deception has failed. It is necessary to clone the real content of a device and produce meaningful data in the event of any interactions. For this reason, we scrape the contents of the file transferring services on devices when developing a profile.

FTPs can require authentication and often do. For certain device models, varying access levels to files is dictated by user authentication. In order to both provide a higher fidelity decoy and assess the tactics of an attacker’s actions it is necessary to include any authentication methods and credentials in the content for FTPs on decoy devices. This provides the necessary functionality to analyze any authentication attempts and keeps the attacker engaged while interacting with the decoy.

3.1.1.4 Network services

Network services are used in IT systems to monitor and manage equipment. Generally, the services most used by OT equipment evolve around time synchronization and log collection. As time synchronization is critical to coordinating the information collected from many sensors and to make the appropriate control actions, time synchronization services are common in OT devices. As electrical system sensors and controllers require high-fidelity time synchronization,

they use services such as the simple network time protocol (SNTP) in broadcast mode or precision time protocol (PTP). To replicate these decoy services, it is either just leaving these ports open or creating real services to connect to network time synchronization services. In addition, some devices provide simple network management protocol (SNMP) access to allow the collection and monitoring of device performance.

3.1.2 Process Control Service Examples

There are a multitude of SCADA and process control protocols used among various industries. These protocols provide functionality to support data collection from distributed sensors and the ability to control the state of the physical process. While there are a lot of protocols used, we have scoped this effort to devices supporting common electrical and building system equipment. The list includes protocols such as Modbus, DNP3, IEC 61850, and BACnet.

Most of the process control protocols define an expansive list of functions and data objects to support a wide range of use cases. As such, most devices do not support all the standard defined functions or objects. To develop decoys, it is necessary to scope down decoy services to only those functions and objects provided by a device profile. For example, a threat that is probing for a specific set of target devices may leverage knowledge of what is and is not supported to figure out when they have found their targets. If a decoy includes or does not include functionality expected in a real device, a target may ignore or bypass the decoy.

Secondarily, as with all protocol standards, there are unique behaviors around unclear or unspecified behaviors of the protocols. For example, in DNP3 you can request a default set of data objects and the response from a device can take many forms. Devices respond in different ways, which helps differentiate from the various vendors equipment. Capturing these unique behaviors and replicating them for decoys is important to deceive threats. An example DNP3 remote terminal unit (RTU) decoy profile is presented in Appendix A**.**

As OT devices are developed for a wide range of use cases, they often exhibit a lot of variety in proprietary features designed for specific use case benefits. Often these proprietary protocols can represent features that are valuable from a threat's perspective and provide high levels of access and control to a device. As such, it is important to mimic these proprietary ports at least at a minimal level. While it may not be known what process is running behind the port or what functionality is provided, it is important to represent the possibility of these ports. As such, these proprietary ports are logged in the profiles and instantiated as decoy ports that allow connection and interaction but do nothing else. Recording the actions taken by threats against these ports will not only inform that a threat is present but may also be informative of the threat's objectives.

3.2 Attivo OT Decoy Virtual Machines

The Attivo BOTsink server is part of the Attivo ThreatDefend platform, architected to detect attackers inside the production networks and can cut attacker in-network loiter time by over 90%. The BOTsink is a physical, virtual, or cloud appliance that converges virtual networking and virtual systems to offer a virtual network that provides a decoy environment to engage with attackers and capture their TTP. The BOTsink server ships with fully licensed operating systems that operators can customize to match the operating environment with a push of a button or replace with their golden images. The system supports Windows workstation images (XP, 7, 8, 10) and Windows Server images (2008, 2012, 2016 & 2019) as well as Linux images (Ubuntu, CentOS, RedHat).

The solution is scalable, and each server can host over 2,000 decoy IPs across hundreds of VLANs/Subnets on layer 2 or layer 3 production, cloud, or datacenter networks. Having the flexibility to run full OS VMs as engagement servers, the BOTsink is fully customizable for authenticity. The system can customize its MAC addresses to match the other system in the subnet and allows the operator to install custom programs such as HMI, Historians, BigData, etc. on the stock VMs or import their own fully customized golden image.

If the system to defend cannot install on standard Linux or Windows VMs, the BOTsink can host emulation of these systems to offer as a target to attackers. Additionally, it can emulate vulnerabilities of such OT/SCADA systems to engage the attacker without these systems being vulnerable themselves.

The ThreatDefend platform can identify early reconnaissance activity and credential harvesting efforts like MiTM attacks to thwart the attacker's efforts from reaching a production system and instead deflects the attack to the decoy environment for further engagement and analysis. The ThreatDefend platform gives high fidelity alerts that cut through the noise and enable the operator to quickly identify attackers and collect troves of highly actionable intelligence to hunt inside the network and eliminate any foothold they might have established.

Figure 1: Attivo Networks SCADA VM 1.0 Services

To demonstrate the high-fidelity decoy services, PNNL collaborated with Attivo Networks to integrate proof of concept implementations into the BOTsink ecosystem. This effort involved the integration of new services into the Attivo decoy SCADA VM. This effort included the creation of 10 high fidelity decoy profiles modeled after real equipment within the PNNL powerNET testbed. Using these profiles, the features, and capabilities of the of the SCADA VM were extended to include realistic web pages, realistic service mappings, and function and objective profiles for SCADA protocols, and new decoy services with realistic responses like telnet configuration

management services. All these features were put together into a deception of a holistic device using the Attivo campaigns capability.

3.3 Attivo OT campaigns

Leveraging the BOTsink capabilities, PNNL and Attivo collaborated to develop out-of-the-box deception campaigns for electrical substations and building markets. These campaigns enable the BOTsink to dynamically match the decoy environment to the operational ones where it is deployed.

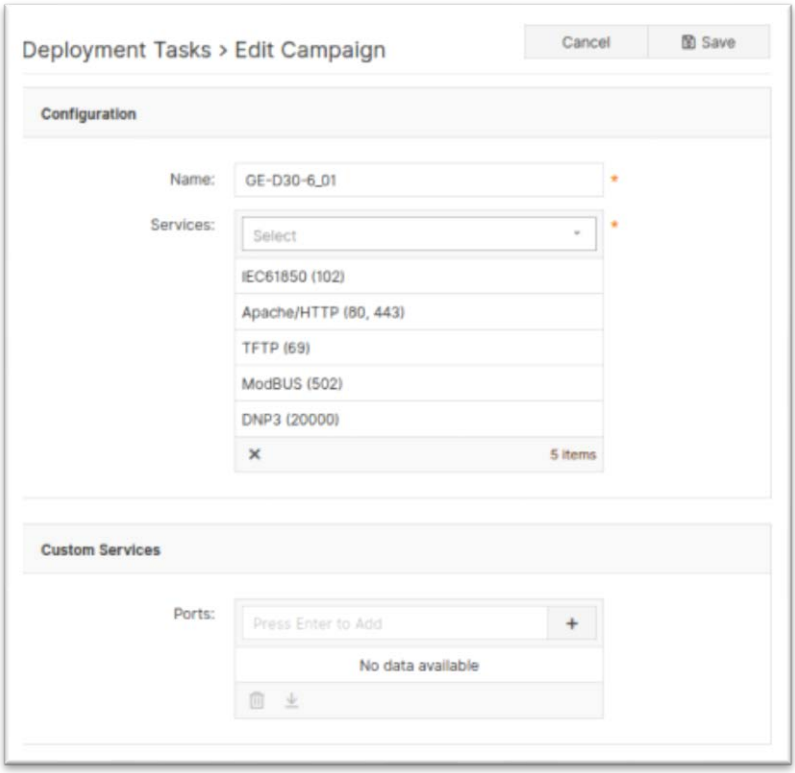


Figure 2: Defining a BOTsink Campaign Template for an OT Device

The campaigns consist of relevant devices like control systems, PLCs, Feeder Protection Relays, Transformer Management Relays, and others, along with the expected protocols like DNP3, Modbus, IEC61850, etc., populated with decoy operational data.

Each campaign can list the types of equipment to construct decoys which include the network services they support and the types of data service functionality and responses they provide. A campaign can encapsulate a single device or multiple depending on the objectives of the deception. An example use case for the flexibility and ways to define a campaign are further discussed in the physics-based simulation subsection of the model driven deception section below.

3.4 Low impact operational safety

Impacts on operations is a high-risk concern when utilities investigate deploying new features like security tools. The cost of sending engineers into the field; sometimes many miles away,

can often be much more than the cost of the security tools themselves. Therefore, not adding risk of increased failure rate outweighs the benefit of increased security and is often preventative of deploying new security tools. Therefore, security that does not impact the performance or failure rate of OT systems are valuable and easier to make the argument to deploy.

Deception technology installs within and around existing systems but not in the middle of any communication paths. Decoys are extra additions to the system that operate in low maintenance modes awaiting a threat to interact with them. Deception defense does not require the modification of any existing system to operate. As deceptive decoys are by their nature ancillary systems that are not necessary to operate a physical process, they are low impact to the performance of the real system that are deployed to protect.

4.0 Model driven deception

Due to the strong integration of real-world physics, OT deception platforms must operate differently than traditional IT deceptions. For instance, turning off a valve will be detected downstream by other sensors because the flow will reduce and stop. Additionally, controllers and applications leverage data from sensors to send control commands to each other. A believable deception must be integrated with the system to project the effects of events. An attack will likely attempt to control the physical process in a negative manner. To make the attacker believe they are achieving their objective, it must predict the effects of these actions, to a reasonable degree.

Traditional cyber deception platforms integrate into a system through the generation of realistic looking data, access/identity control accounts, and/or honey tokens and the deployment of decoy services such as web applications, file shares, or remote access. Each of these components can be made to appear as part of the system with realistic banners, directory, and filenames related to the business or corporate branding. This data can be distributed onto real workstations and servers to be found by attackers and direct them to the decoy services. These capabilities are useful and could be used within cyber-physical deceptions, but they alone are insufficient.

The threat model driving this research is an adversary knowledgeable in cyber-physical systems with a targeted objective to affect the operation of the physical process. Under this threat model the attacker will be aware of, and searching for, specific categories of equipment to understand how they interface with the physical process and how they can be controlled or manipulated. The objective of this research is to delay the attack from successful completion while increasing the probability of detecting the threats presence and actions. Achieving these goals under this threat model requires sufficient fidelity.

To provide enough fidelity there are additional requirements that must be met by cyber-physical deception platforms. First, the deception platform must provide the ability to simulate a model of the physics of a real system process. This includes supporting the ability to generate realistic variable data from decoy device responses. Second, a deception platform must provide the ability to define new devices and connect them to the physics model being simulated. We will go into more depth about the attributes of these decoy devices later. Finally, the decoys should appear as tempting, easy to exploit targets, that are part of the real operating cyber-physical system. We propose that these three requirements are necessary for an adequate deception of cyber-physical decoys in a real system.

4.1 Types of Deception

There are multiple ways in which a model driven deception can be deployed. These include cloning, copies, and integrated deceptions. Each type of deception is useful under different circumstances. Their use and utility are a function of the location of the deception in relation to the real system and what type of threat is being countered. A brief overview of each type of deception and microgrid examples of how they could be utilized are provided.

4.1.1 Clone

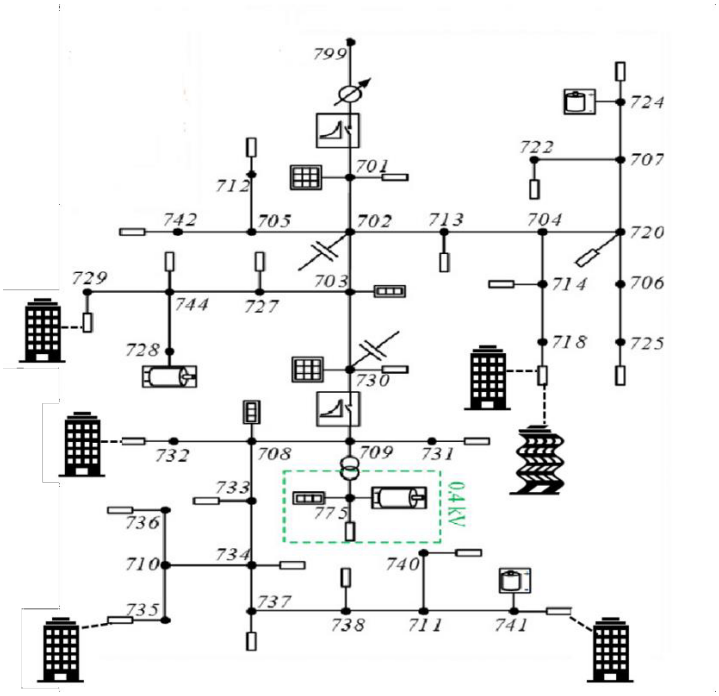


Figure 3: Decoy clone of existing system

A clone deception is when an exact replica is presented as the real system to deceive an attacker that they are interacting with the real system components. This type of deception traps the attacker into a fictional world that is directly related to the real system. The model for this type of deception can be driven directly from observed data of the real system. Only upon control or other altering interactions by the attacker is the projection of effect necessary. In a microgrid, a deceptive clone could be developed that mimics an entire building HVAC cyber-physical system as shown in Figure 3. The clone, represented by the mirage building, is posing as the adjacent building in this deception. In this type of deception there would be some mechanism, like a firewall, IPS, or VPN, that would determine when to send connections to the cloned system instead of the real system.

4.1.2 Copy

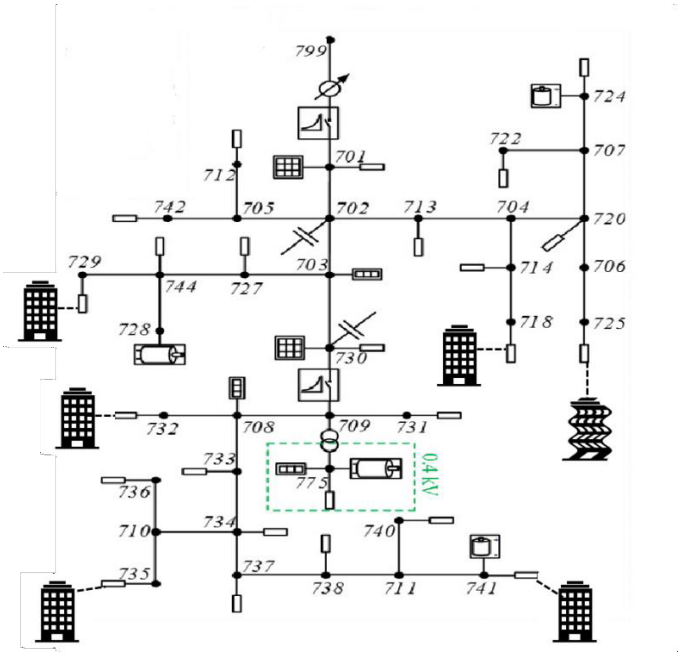


Figure 4: Decoy copy of additional system

A copy deception is like the clone where replicas of a real system are presented. However, the difference with this deception is that one or more replicas are presented within the same network perspective as the real system. This type of deception makes it appear to observers that there are multiple running cyber-physical systems and provides an obfuscation style defense where an attacker must determine which system is the real system. Each decoy copy interacts with a simulation of the real system model and can be driven at offset times or with some form of data fuzzing such that the data does not appear to be exactly the same, further obfuscating which system is the correct system. Each copy can respond and react to interactions independently.

An example of this type of deception in our microgrid example would be to create a copy, or multiple copies of a building on the microgrid and deploy deceptive ones in different locations. The copied building could have the same components as the original or differences to add to the confusion.

4.1.3 Integrated Decoys

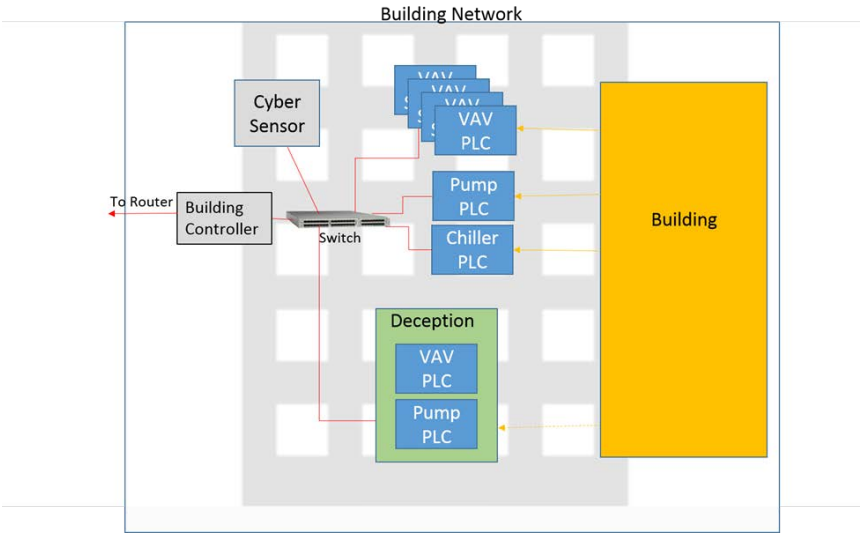


Figure 5: Integrated decoy of new system

Integrated deceptions places newly conceived decoys within the real system where the decoy modeled data is defined such that it logically relates to real data within the system. For example, in a chemical process, a decoy could be made that controls a fictional valve downstream that controls the flow to a decoy sensor. The modeled values output from these decoys are extrapolated or derived as some function of the real system values. Through these extrapolations it appears that the decoy produces data related through physics to the real system and thereby providing a high-fidelity target of value to an attacker that has already bypassed other defenses and infiltrated the OT network. Figure 5 depicts integrate decoys for PLCs controlling new variable air volume (VAV) and chiller pumping systems for an additional structure, like a floor or zone, related to the real building HVAC systems.

In all three deception types, it is necessary to have a model of a real system to drive the behavior and responses of the deceptions to make them appear realistic and relevant. There are two general methods for creating a model and driving a simulation to feed realistic data to decoys: data driven and physics-based models.

4.2 Physics based simulation

The first type of model driven deception is physics-based models. Physics based models incorporate mathematical representations of the physics behind any given physical process. Through these mathematical models it is possible to simulate the behavior of physical processes under various conditions. Utilizing physics-based models it is possible to develop some common pre-defined system models that can simulate and generate data to drive variations of decoy deployments around different domains.

An example pre-canned model would be an electrical distribution substation. Within distribution substations are electrical feeders which are the systems that provide the final leg of distribution to houses and businesses. While every utility can construct their feeders in whatever configuration they desire, when creating a pre-canned example, it is desirable to select a broadly applicable case such as prototypical feeders (Schneider et. al. 2018). Prototypical feeders are models of commonly reoccurring features from a survey of the United States distribution utilities. Figure 6 displays a generic one-line diagram for a feeder that shows the

flow of electricity from the transmission system on the left, through a transformer to lower the voltage to safe levels for distributing to consumers, multiple electrical safety (like breakers and reclosers to cut power to the segment) and conditioning (like tap changers and capacitors for managing stable voltages) devices, and out to the circuit of power lines to neighborhoods and commercial buildings.

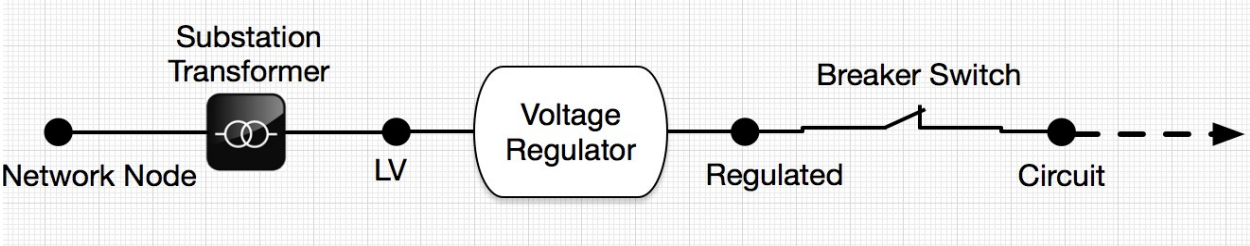


Figure 6: Generic feeder one-line diagram

Most of the components for an electrical feeder are housed with a substation yard. Within a substation there are sensors and controllers connected to the feeder to monitor and manage the safe and reliable flow of electricity. Supervisory control and data acquisition (SCADA) are the communication architecture through which data is collected from local devices and communicated to one or more supervisory systems controlled by operators. At the top of the hierarchical SCADA communication architecture is a control center where operators and dispatchers monitor all or large areas for a distribution utility's assets. Each substation generally has a communication gateway called an RTU for aggregating all the information from the substation and provides it back to the central control room. Within the substation, the RTU aggregates data from multiple intelligent electronic devices (IEDs) such as meters and relays that interface with physical substation components like transformers and switches. Figure 7 depicts a general SCADA architecture for a distribution utility. The DNP3 is the most common SCADA communication protocol currently used within the United States. Other protocols are prominent in other locals like IEC 61850 and 60870-5 in Europe.

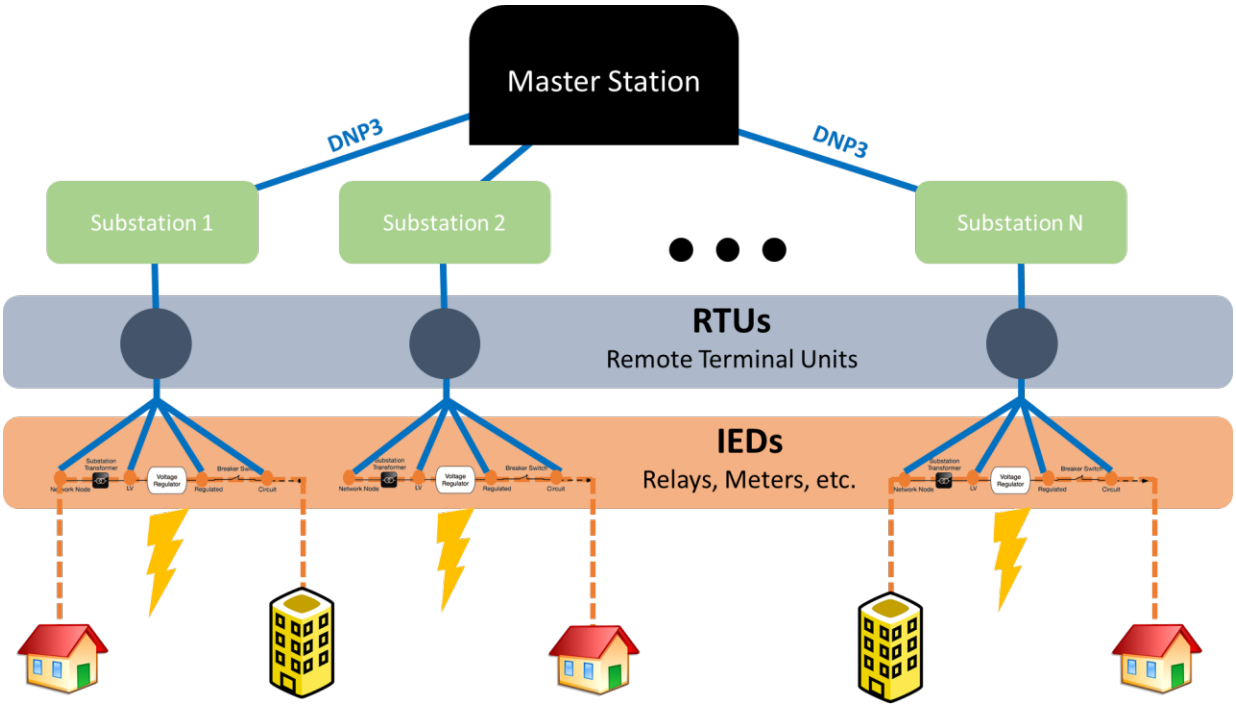


Figure 7: SCADA communication hierarchy

Based on the specific feeder construction there is some number of inputs/outputs (I/O) present for monitoring and controlling all the components in the substation. For example, the R2-1247-2 prototypical feeder has 26 analog input and 6 output variables and 51 binary input and output variables. These variables are those that are available to be mapped to the decoy devices.

These physics-based models enable deployment of different decoy configurations based on a user's needs including the number of decoys, configuring which protocols are used, and the mapping of variables to protocol points in each decoy. The number of decoys is configurable bounded by the number of logically separable pieces but in real systems a single relay can monitor many points and provide multiple functions like metering, reclosing, and protection. The minimum and easiest decoy campaign possible with these feeder models is a single RTU that represents the communication gateway to the substation and all variables are mapped to that single unit. This configuration is common in distribution substations where the communication between the IEDs is all legacy serial communication and wired analog signals and the long-distance communication between the RTU and the control room is Ethernet based. The most complex campaign scenario is to map all of the variables to relays such as a high side relay to break the current coming from the sub-transmission system, a feeder overcurrent protection relay, a differential relay on the transformer, and some number of recloser relays mid-circuit. A decoy RTU can be created to poll all the decoy relays over an Ethernet network to make the substation environment appear active and operational. An example listing of variables from a prototypical feeder substation is provided in Appendix B**.**

Once a mapping of data for a campaign is complete, it is possible to execute the decoys. The PNNL developed GridLAB-D simulator is used to execute the prototypical feeder models. Through wrapper code the values for the variables are captured and inserted into a database every simulation cycle, such as every second. The database acts as a persistent memory of what is happening within simulated system. With a real time updating state of the system it is then possible to leverage the campaign decoy mappings to update the decoy data states from

the database to present a high fidelity and realistic set of data to the network and to active threats probing the system. If a threat discovers a point within a decoy that it would like to control, like a switch to cut power to part of the whole feeder, the database also provides a data path to send state changes from SCADA protocol control commands to state changes in the simulation. The wrapper code also checks for requested state changes every cycle and pushes them to the GridLAB-D simulation to alter the system state and generate resultant data expected, such as 0 voltage downstream of the switch that was opened. Now a threat observing the decoys responses will believe that their attack was successful, and they have achieved their goals.

4.3 Data driven simulation

While physics-based simulations are powerful tools able to produce highly accurate results to enable high fidelity of decoy response, the development of physics-based models is resource intensive requiring domain expertise. This limits the ability to develop custom built models that generalize across and within industries and fit a specific organization's unique characteristics. Therefore, another solution is necessary to enable highly tuned OT deception campaigns. We have been investigating and developing a data driven approach using machine learning (ML) to generate "good enough" models of operational OT systems.

ML has been successfully implemented in many different areas in recent years. ML applications range from handwriting recognition, image classification, self-driving cars, and more. Many of the algorithms and techniques used in ML were developed in the late 1990's and early 2000's. But recently, because of the advances in hardware (CPU'S, TPU's, GPU's, etc.), we have the computational power to implement these algorithms. The field of ML keeps evolving at an ever-faster pace and new techniques and areas of application are being discovered. In this work we explored the application of ML in cyber-deception. It is not only computers and phones that are connected to the internet or a cyber-network. Our approached investigated the use of Long-Short Term Memory network (LSTM) and a novel shallow neural network Equation Learner (EQL) (Sahoo, Lambert, and Martius 2018), on surrogate time series data representing the outputs of different building sensors.

Data collected from an operational system, from methods like network SPAN ports or data historians, can be used with these ML approaches to generate models and simulate their responses under different states. These models can be used just like the previous physics models to create the data to drive realistic responses of decoys. However, generating models from collected data presents a challenge in creating accurate models because the data collected will likely not provide data on all variables incorporating the full system dynamics. The goal of our machine learning solution is to generate "good enough" to deceive threats long enough, minutes to hours, to provide incident responders enough time to take mitigative actions.

To find the most effective method we have studied two ML approaches; a common recurrent neural network approach and a newer cutting-edge extrapolative approach. A building control system test case was used to compare the performance of the two approaches and highlight which methodology is better suited to generate sufficient models for simulating realistic responses. The rest of this section discusses the results of this study.

4.3.1 Surrogate Time Series Sensor Data Description

To study the performance of the two ML approaches it is necessary to generate data for training, validating, and testing. The surrogate time series data was obtained using the software package Modelica (University of California n.d). Modelica is an open source software package developed by people in the Lawrence Berkley National Laboratory. This software can model dynamic systems for buildings and energy control systems. For the purposes of this work we use the 5 parameters in Table 1 below for the monitoring and control of water temperature for a building cooling system.

Feature Variables
Chilled Water Flow Rate
Chilled Water Transfer Rate
Chilled Water Return Temperature
Chilled Water Supply Temperature
Chilled Water Valve Position

Table 1: Feature variables.

The last entry in the time series is used to predict the next value. These values are simulated to be recorded every second using weather data recorded from March-July 2018. We used the data from March-May for training, June for validation, and July data is used for testing and comparison purposes.

4.3.2 Long-Short Term Memory Approach

LSTM's are a subset of what are known as Recurrent Neural Networks (RNN). LSTM are very useful when trying to model processes where the next value in the series depends on previous ones. Our network consisted of a LSTM layer with 1024 units connected to four Dense layers with 512, 256, 256, 5 neurons, respectively. Figure 8 depicts the design of the network.

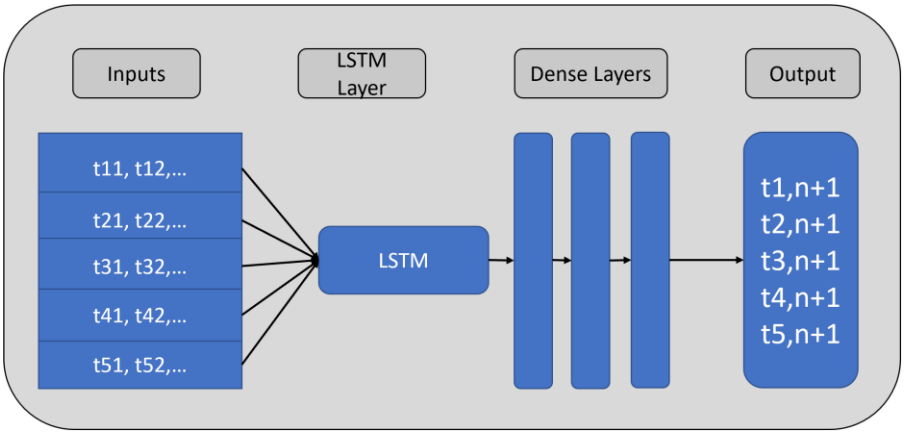


Figure 8: Graphical representation of the LSTM based network.

Table 2 shows the training times and corresponding validation losses at the end of 25 epochs.

Sample Size	Time	Val. Loss
44,359	3.69	0.001400
87,030	7.02	0.001300
122,426	8.494	0.001400
165,625	11.421	0.002100

Table 2: LSTM Based Net Training Times and Validation Losses

The following graphs in Figure 9 and 10 show the true, versus the predicted, values and the point wise error on the test data. The point wise error is defined as the true value minus the prediction at time t0. Thus, a perfect prediction would be a straight line at x=0.

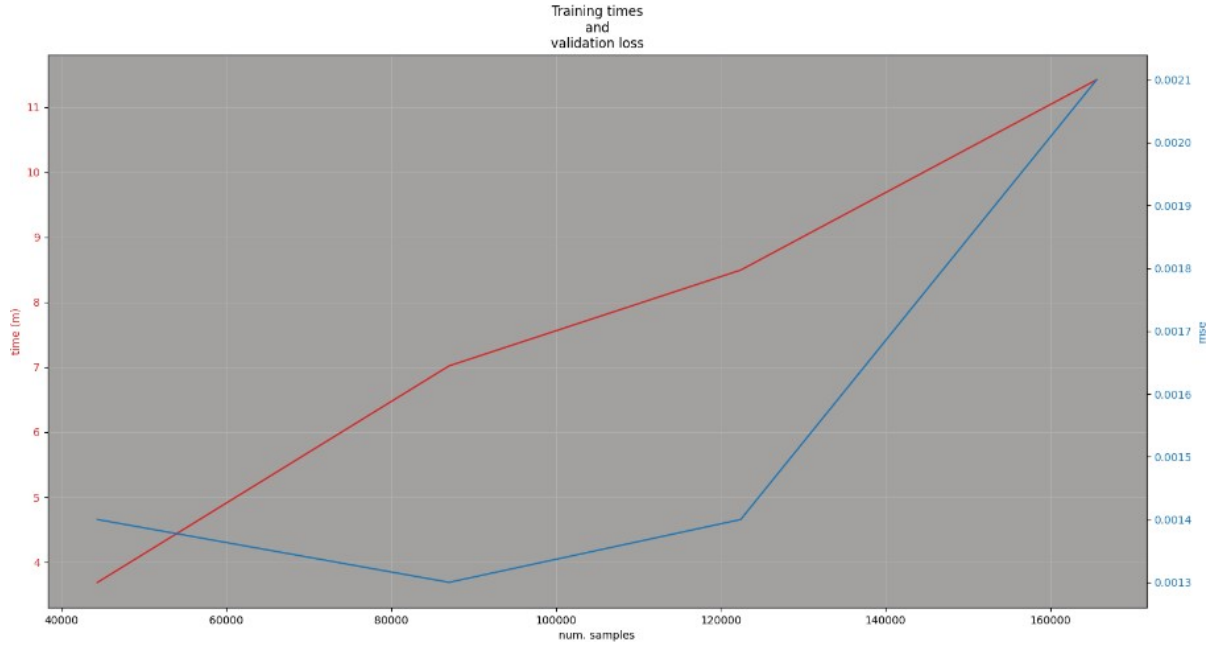


Figure 9: LSTM training times and validations losses.

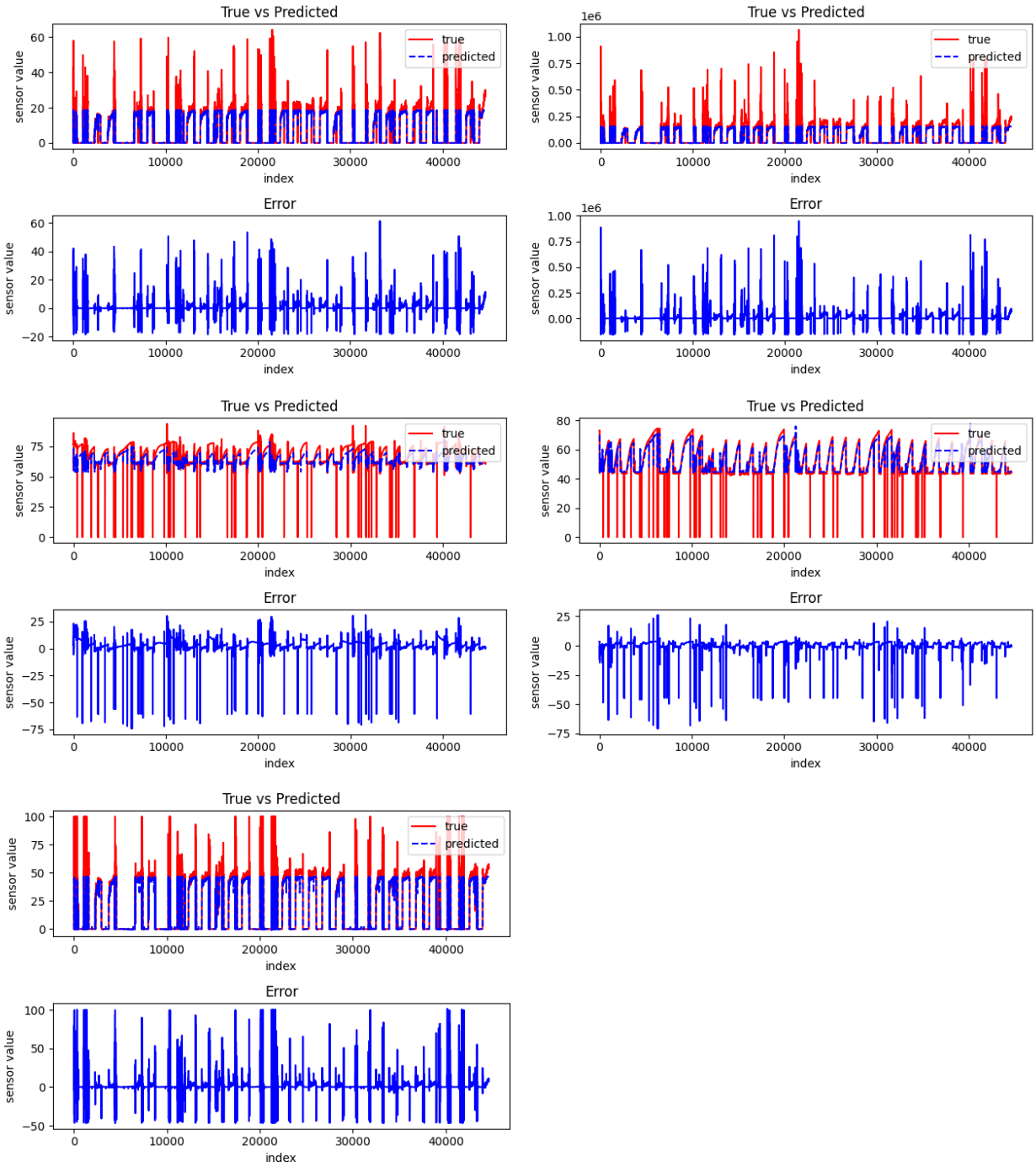


Figure 10: Plots for the 5 different Chilled Water variables.

4.3.3 Equation Learner Approach

This method was developed by Subham Sahoo, Christoph Lampert, and Georg Martius (2018). This is a shallow neural network that tries to learn the underlying mathematical equation(s) that best represents the data. Each node in the network represents a mathematical operation (e.g. addition, subtraction, etc.). The advantage of this method over other equation learners is that it overcomes the obstacles to be able to learn the division operation. The same training,

validation, and test data splits were used as in the LSTM approach. Table 3 shows the training times and validation loss at the end of 25 epochs.

Sample Size	Time	Val. Loss
44359	0.864	0.017
87030	1.29	0.022
122426	1.638	0.015
165625	2.158	0.026

Table 3: Training times and validation error at the end of 25 epochs for the EQL base network.

The following graphs in Figures 11 and 12 show EQL's predictions versus the test data. The point wise error is shown as well.

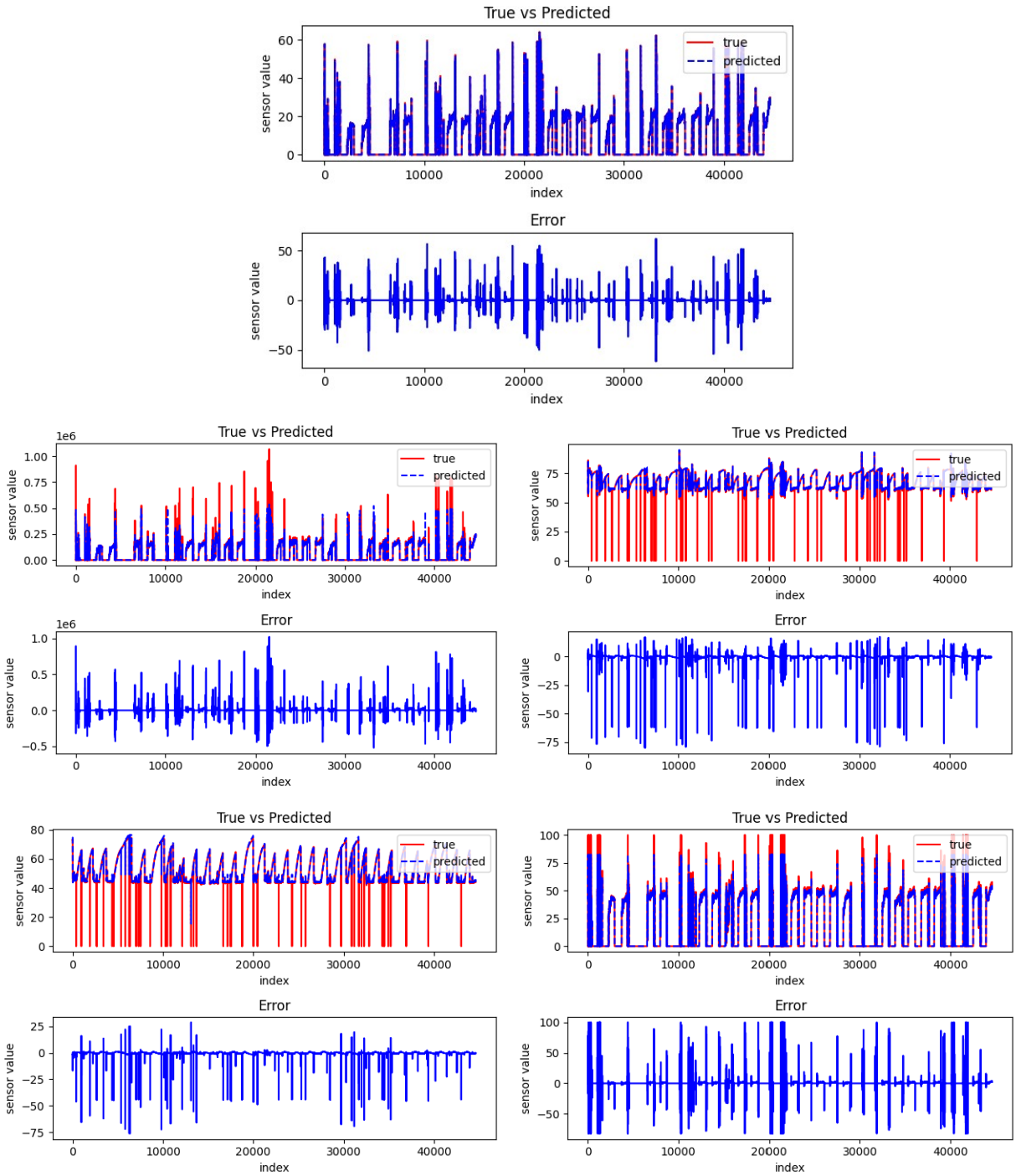


Figure 11: EQL's predicted values against the true values.

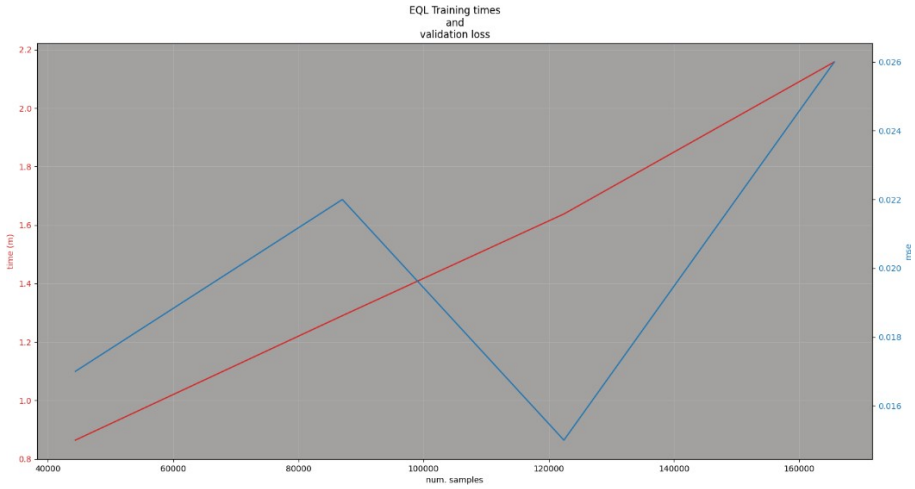


Figure 12: EQL's based training times and validation error.

Like we have described above this method generates the equations that describe the data and can make a graph of these equations using the software package Graphviz. An example graphing is shown in Figure 13 below.

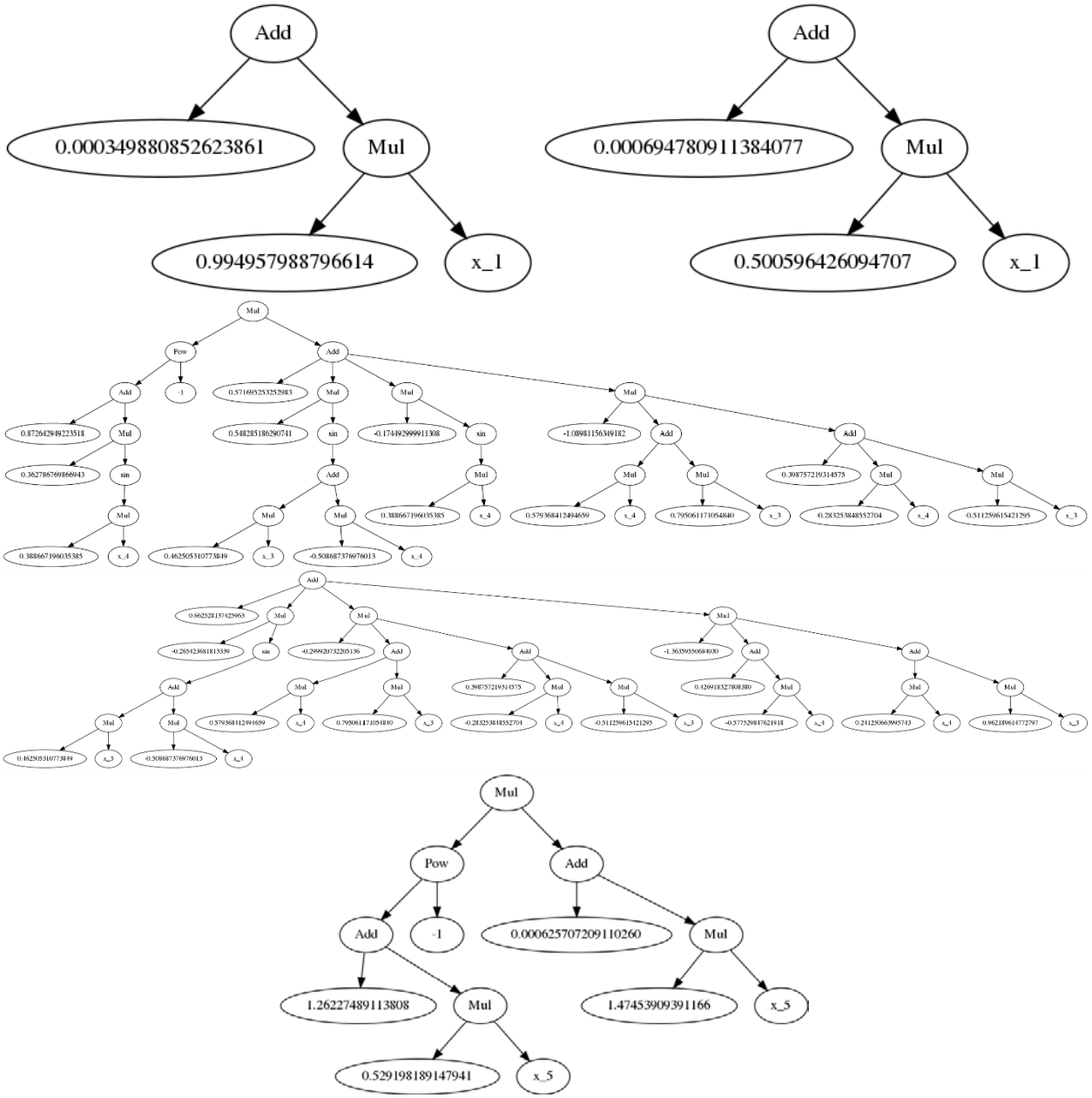


Figure 11: Equations learned by the EQL algorithm. Each equation corresponds to the variables in table 1, respectively.

4.3.4 LSTM vs EQL

In general, the EQL method was able to do better predictions than the LSTM; moreover, it also trained faster. However, the EQL method required more training epochs to achieve a desired validation loss. The following graphs in Figure 14 show the comparison of the predicted values of both approaches versus the test data.

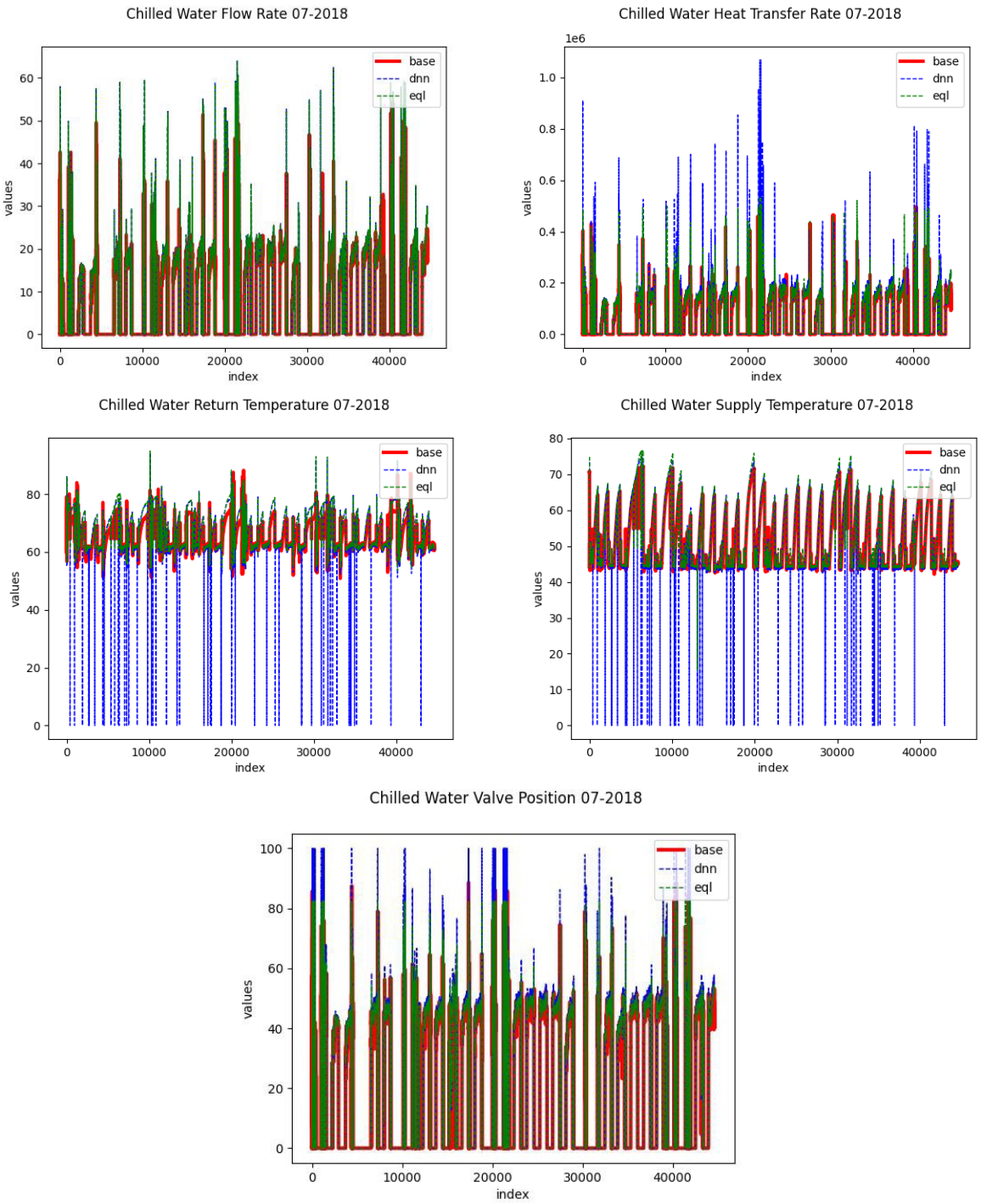


Figure 12: LSTM and EQL predicted values versus test data

We can observe graphically that in general the EQL net had more success in capturing the test data features. It seemed that the LSTM had a hard time predicting the spikes on the data. Some of these spikes seem like an artifact since we are using surrogate/generated data. Future work could benefit from using real world data for training. However, both methods had relative success in demonstrating that we can generate sensor, time series data. The EQL method was more successful at capturing the complex features of a chilled water return system. Moreover,

since the EQL generates the equations these are better suited to be run in a loop unlike the LSTM that required corrections after some time steps.

4.4 Simulation performance

The resources utilized to run simulations of the physics of an OT system take away from the pool available to run decoys. Therefore, it is important to understand how much resources are necessary to execute the model simulations and how they scale with decoys. In this section we present the results evaluating the performance of the two model driven approaches.

4.4.1 Physics based simulation performance

Understanding the performance of the simulation tools as they operate is crucial to understanding their scalability. We performed some performance testing to characterize the amount of resources; CPU and memory, necessary to run a model. Because the physics-based simulations include much more complex integrated dynamics cancelations than the learned models; in this section we focus on the performance of the physics-based models to provide an expectation for an upper bound of performance. Figure 13 shows the performance results of executing the model described in our use case above.

	CPU %			RAM %		
Application	Mean	Min	Max	Mean	Min	Max
python	13.5	8.0	21.9	0.4	0.4	0.4
GridLAB-D	0.0	0.0	0.0	9.6	9.6	9.6
FNCS broker	0.3	0.0	1.0	0.1	0.1	0.1

Figure 13: Physics based model simulation performance

4.4.2

Three programs are involved with running the physics-based model simulation. The GridLAB-D application runs the actual simulation of the distribution feeder. The FNCS broker is a middleware communication platform that enables the exchange of data with the running GridLAB-D application. Finally, a python wrapper provides the code to collect and send data to the FNCS broker and push and pull data from the database. As you can see, in aggregate the amount of resources necessary to run all these applications is very minimal. The python wrapper requires the most resource utilization because it is constantly active polling for data from the database and the message bus. The FNCS broker consistently uses a small amount of resources to exchange the small amount of data involved. Finally, the GridLAB-D does not utilize much CPU at all and maintains a consist amount of memory based on the model being simulated. The 0.0 CPU utilization is believed to be an artifact of our data collection methodology in that samples were only taken every 3 seconds and the spikes of CPU usage for GridLAB-D are brief. The reason for this is because we are operating GridLAB-D in real-time mode, so it only must calculate the system state every second which is fast and then it waits until the next state calculation. Therefore, GridLAB-D is waiting most of the time leading to a very low CPU utilization.

The process we used to calculate this data was to utilize the Linux top utility in batch mode which samples the CPU and RAM utilization every 3 seconds. We sampled for 160 cycles to generate the data in the table. The system specifications for this performance study are as follows:

Processor: Intel® Core™ i7-6920HQ CPU @ 2.90GHz 2.90GHz
RAM: 32.0 GB

4.4.3 Data driven simulation performance

Running the simulations in real time requires minimal resources as can be seen from the physics-based simulation performance results. However, the learning process for the data driven approach is also potentially an expensive process. As each scenario will require a unique number of samples to adequately train a model, we ran the two ML approaches with 4 different sample sizes to discover the resource utilization range and averages listed in Table 4 and Table 5. All the tests were performed on the same system, the specifications of which are detailed in Table 6.

As can be seen, and in staying consistent with our previous results, the EQL based learning method was more efficient across the board when compared with the LSTM based learning method. The LSTM displays more variability in CPU utilization (over 100% CPU utilization means using additional processor cores) and is on approximately 2x as processor intensive compared to EQL. Both processes increase in memory utilization as the sample size increases which makes sense because they must hold more data in memory. The minimum percentages are all fairly consist which likely represents the actual processing memory requirement not including a data storage for samples during processing. The resident memory follows closely to the memory percent utilization and the virtual memory stays consistent across all tests for both approaches. EQL is overall less memory intensive than LSTM.

Table 4: EQL resource utilization when learning per sample size

	EQL Based											
	CPU %			MEM %			VIRTUAL (M)			RESIDENT (M)		
Sample Size	mean	max	min	mean	max	min	mean	max	min	mean	max	min
44,359	84.04	175.00	7.30	5.80	6.00	5.10	3009.73	3033.78	2986.36	467.71	485.57	410.66
87,030	143.11	177.10	41.50	6.45	6.90	5.80	3043.43	3065.03	3019.98	517.71	555.05	461.97
122,426	145.11	176.80	67.10	7.05	7.60	5.60	3078.19	3101.58	3017.38	566.70	606.52	449.20
165,625	143.77	168.40	66.40	7.47	8.10	5.40	3105.03	3156.36	2803.58	600.02	647.14	430.75

Table 5: LSTM resource utilization when learning based on differing amounts of samples

	LSTM Based											
	CPU %			MEM %			VIRTUAL (M)			RESIDENT (M)		
Sample Size	mean	max	min	mean	max	min	mean	max	min	mean	max	min
44,359	235.14	285.80	110.90	8.76	10.00	7.70	3116.43	3240.88	3044.27	705.64	800.45	621.42
87,030	279.80	309.60	205.00	8.52	9.80	7.90	3109.39	3267.69	3038.06	682.74	785.49	632.77
122,426	254.95	311.00	161.50	8.92	10.40	7.90	3126.97	3238.54	3041.12	718.75	831.41	634.78
165,625	245.42	313.60	139.20	9.20	10.20	7.50	3160.83	3275.38	3045.49	738.06	819.03	600.98

Table 6: Experiment setup for data driven performance tests

lscpu	
CPU(S)	4.00
Thread(s) per core:	1.00
Core(s) per socket:	4.00
Model name:	Inter(R) Core (TM) i5-7300HQ CPU @ 2.50GHz
CPU MHz:	900.12
lsmem	
Memory block size:	128M
Total online memory:	8G
Total offline memory:	0B
dmidecode --type memory	
Total Width:	64 bits
Size:	8192 MB
Type:	DDR4
Type Detail:	Synchronous
Speed:	2400 MT/s

5.0 Conclusion

Deception defense is a useful solution to ensnare and abate active threats activities. Deception can be a solution to help our critical infrastructure detect and respond faster to ongoing cyber-attacks against their most critical systems. However, as we have shown, traditional IT deception techniques need to be enhanced and extended to create real enough decoys for defense of cyber-physical systems. With profiles of services for high fidelity mimicking of embedded equipment and simulated models of physical systems to drive their responses, deception campaigns can be crafted that are enticing and convincing.

A deception defense solution for OT environments provides a flexible capability that enables users to customize their defenses based on their current system configuration and threat tactics to increase the likelihood of detecting active attacks. All these capabilities are possible without the potential to increase system mean time to failure that other security solutions do. Cyber deceptions are instantiated amidst and around operational systems but do not occlude their communication or performance.

The R&D outcomes of increasing decoy fidelity and creating a process for model driven deception from this project move the bar on how cyber deception can be utilized for critical infrastructure defense. We have demonstrated proof of concept capabilities to highlight the value of these capabilities. Through performance analysis we have shown that these approaches scale and can be utilized within the existing Attivo Networks BOTsink platform.

6.0 References

Antonioli, D. and N. O. Tippenhauer, 2015. “Minicps: A toolkit for security research on cps networks,” in Proceedings of the First ACM Workshop on Cyber-Physical Systems-Security and/or Privacy, ser. CPS-SPC ’15, 2015, pp. 91–100.

Cohen, Fred et al. 2001. A Framework for Deception. http://all.net/journal/deception/Framework/Framework.html.

Cyber.nj.gov. 2017. “Stuxnet - NJCCIC Threat Profile.”. www.cyber.nj.gov/threat-center/threat-profiles/ics-malware-variants/stuxnet.

Greenberg, A. 2018. The Untold Story of NotPetya, the Most Devastating Cyberattack in History. https://www.wired.com/story/notpetya-cyberattack-ukraine-russia-code-crashed-the-world/.

Hartmann, Wayne. 2019. Generator Protection Theory & Application. https://na.eventscloud.com/file_uploads/529211ae18f0798b5a92f1ced9ac3673_WSU_GENTheory_1PageperSlide_160114.pdf

Honeynet Project. 2019. The Honeynet Project. https://www.honeynet.org/

Honeywell. 2018. Honeywell Industrial USB Threat Report: Universal Serial Bus (USB) threat vector trends and implications for industrial operators.

Lee, Robert M., Michael J. Assante, and Tim Conway. 2016. Analysis of the Cyber-attack on the Ukrainian Power Grid. https://ics.sans.org/media/E-ISAC_SANS_Ukraine_DUC_5.pdf

Litchfield, S. et al. 2016. “Rethinking the Honeypot for Cyber-Physical Systems”. *IEEE Internet Computing* 20.5: 9–17. DOI: 10.1109/MIC.2016.103.

Pingree, Lawrence. 2015. Competitive Landscape: Distributed Deception Platforms. https://www.gartner.com/en/documents/3402317.

Pothamsetty, Venkat and Matthew Franz. n.d. SCADA HoneyNet Project: Building Honeypots for Industrial Networks. http://scadahoneynet.sourceforge.net/.

Rist, Lucas et al. 2018. Conpot: ICS/SCADA Honeypot. http://conpot.org

Sahoo, Subham, Christoph Lampert, Georg Martius . 2018. Learning Equations for Extrapolation and Control. *Proceedings of the 35th International Conference on Machine Learning*

Sahoo, Subham, Christoph Lampert, Georg Martius; 2018. Learning Equations for Extrapolation and Control. https://github.com/martius-lab/EQL_Tensorflow

Schneider, Kevin P, Yousu Chen, David P Chassin, Robert G Pratt, David W Engel, and Sandra E. Thompson. 2008. Modern Grid Initiative Distribution Taxonomy Final Report.

Simoes, Paulo et al. 2013. “On the use of Honeypots for Detecting Cyber Attacks on Industrial Control Networks”

Stoll, Clifford. 1989. The Cuckoo's Egg: Tracking a Spy Through the Maze of Computer Espionage.

Stouffer, Keith A. et al. 2011. SP 800-82 Rev 2. Guide to Industrial Control Systems (ICS) Security: SupervisoryControl and Data Acquisition (SCADA) Systems, Distributed Control Systems (DCS), and Other Control System Configurations Such As Programmable Logic Controllers (PLC).

Subbarao, Kris et al. 2013. Transactive Control and Coordination of Distributed Assets for Ancillary Services.

University of California. n. d. Modelica Buildings Library. https://simulationresearch.lbl.gov/modelica/index.html

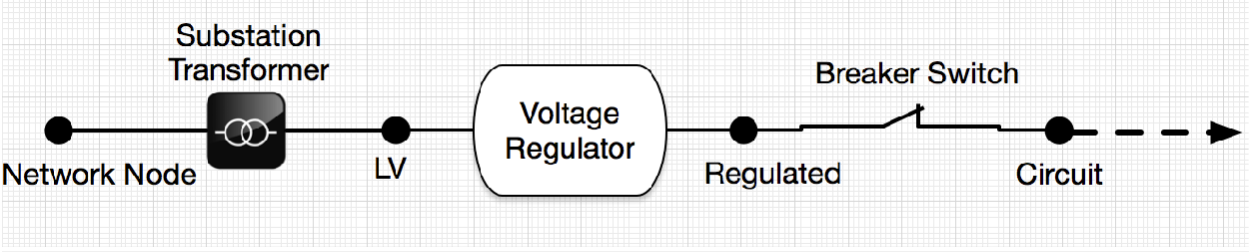
Washington State University. n.d. Generator Protection Theory & Application. https://www.eiseverywhere.com/file_uploads/529211ae18f0798b5a92f1ced9ac3673_WSU_GENTheory_1PageperSlide_160114.pdf.

Wilhoit, Kyle and Stephen Hilt. 2015. The GasPot Experiment: Unexamined Perils in Using Gas-Tank-Monitoring Systems. http://conpot.org/.

Younis, O. and N. Moayeri. 2017. "Employing Cyber-Physical Systems: Dynamic Traffic Light Control at Road Intersections". *IEEE Internet of Things Journal* 4.6: 2286–2296. DOI: 10.1109/JIOT.2017.2765243.

Appendix A – Example DNP3 variables from R2-1247-2 prototypical feeder model

The variables in the table below all map to this generic picture of feeder components.



Feeder Object Name	Prototypical Feeder Property	DNP3 Parameter Type	Notes
Network Node	voltage_A_mag	analog input	
Network Node	voltage_B_mag	analog input	
Network Node	voltage_C_mag	analog input	
Substation Transformer	current_A_mag	analog input	
Substation Transformer	current_B_mag	analog input	
Substation Transformer	current_C_mag	analog input	
Substation Transformer	power_out_real	analog input	
Substation Transformer	power_out_imag	analog input	
Substation Transformer	power_out_A_real	analog input	
Substation Transformer	power_out_B_real	analog input	
Substation Transformer	power_out_C_real	analog input	
Substation Transformer	power_out_A_imag	analog input	
Substation Transformer	power_out_B_imag	analog input	
Substation Transformer	power_out_C_imag	analog input	
LV	voltage_A_mag	analog input	
LV	voltage_B_mag	analog input	
LV	voltage_C_mag	analog input	
Regulated	voltage_A_mag	analog input	
Regulated	voltage_B_mag	analog input	
Regulated	voltage_C_mag	analog input	

Regulator 2 Up-stream	voltage_A_mag	analog input	Regulator is located mid-circuit and is not represented in diagram
Regulator 2 Up-stream	voltage_B_mag	analog input	Regulator is located mid-circuit and is not represented in diagram
Regulator 2 Up-stream	voltage_C_mag	analog input	Regulator is located mid-circuit and is not represented in diagram
Regulator 2 Down-stream	voltage_A_mag	analog input	Regulator is located mid-circuit and is not represented in diagram
Regulator 2 Down-stream	voltage_B_mag	analog input	Regulator is located mid-circuit and is not represented in diagram
Regulator 2 Down-stream	voltage_C_mag	analog input	Regulator is located mid-circuit and is not represented in diagram
Voltage Regulator	tap_pos_A	analog input and output	
Voltage Regulator	tap_pos_B	analog input and output	
Voltage Regulator	tap_pos_C	analog input and output	
Regulator 2	tap_pos_A	analog input and output	Regulator is located mid-circuit and is not represented in diagram
Regulator 2	tap_pos_B	analog input and output	Regulator is located mid-circuit and is not represented in diagram
Regulator 2	tap_pos_C	analog input and output	Regulator is located mid-circuit and is not represented in diagram
Breaker Switch	phase_A_state	binary input and output	
Breaker Switch	phase_B_state	binary input and output	
Breaker Switch	phase_C_state	binary input and output	
Capacitor 1	switchA	binary input and output	Capacitor is located mid-circuit and is not represented in diagram
Capacitor 1	switchB	binary input and output	Capacitor is located mid-circuit and is not represented in diagram
Capacitor 1	switchC	binary input and output	Capacitor is located mid-circuit and is not represented in diagram
Capacitor 2	switchA	binary input and output	Capacitor is located mid-circuit and is not represented in diagram
Capacitor 2	switchB	binary input and output	Capacitor is located mid-circuit and is not represented in diagram
Capacitor 2	switchC	binary input and output	Capacitor is located mid-circuit and is not represented in diagram
Capacitor 3	switchA	binary input and output	Capacitor is located mid-circuit and is not represented in diagram
Capacitor 3	switchB	binary input and output	Capacitor is located mid-circuit and is not represented in diagram
Capacitor 3	switchC	binary input and output	Capacitor is located mid-circuit and is not represented in diagram
Capacitor 4	switchA	binary input and output	Capacitor is located mid-circuit and is not represented in diagram
Capacitor 4	switchB	binary input and output	Capacitor is located mid-circuit and is not represented in diagram
Capacitor 4	switchC	binary input and output	Capacitor is located mid-circuit and is not represented in diagram

Switch 1	phase_A_state	binary input and output	Switch is located mid-circuit and is not represented in diagram
Switch 1	phase_B_state	binary input and output	Switch is located mid-circuit and is not represented in diagram
Switch 1	phase_C_state	binary input and output	Switch is located mid-circuit and is not represented in diagram
Switch 2	phase_A_state	binary input and output	Switch is located mid-circuit and is not represented in diagram
Switch 2	phase_B_state	binary input and output	Switch is located mid-circuit and is not represented in diagram
Switch 2	phase_C_state	binary input and output	Switch is located mid-circuit and is not represented in diagram
Switch 3	phase_A_state	binary input and output	Switch is located mid-circuit and is not represented in diagram
Switch 3	phase_B_state	binary input and output	Switch is located mid-circuit and is not represented in diagram
Switch 3	phase_C_state	binary input and output	Switch is located mid-circuit and is not represented in diagram
Switch 4	phase_A_state	binary input and output	Switch is located mid-circuit and is not represented in diagram
Switch 4	phase_B_state	binary input and output	Switch is located mid-circuit and is not represented in diagram
Switch 4	phase_C_state	binary input and output	Switch is located mid-circuit and is not represented in diagram
Switch 5	phase_A_state	binary input and output	Switch is located mid-circuit and is not represented in diagram
Switch 5	phase_B_state	binary input and output	Switch is located mid-circuit and is not represented in diagram
Switch 5	phase_C_state	binary input and output	Switch is located mid-circuit and is not represented in diagram
Switch 6	phase_A_state	binary input and output	Switch is located mid-circuit and is not represented in diagram
Switch 6	phase_B_state	binary input and output	Switch is located mid-circuit and is not represented in diagram
Switch 6	phase_C_state	binary input and output	Switch is located mid-circuit and is not represented in diagram
Switch 7	phase_A_state	binary input and output	Switch is located mid-circuit and is not represented in diagram
Switch 7	phase_B_state	binary input and output	Switch is located mid-circuit and is not represented in diagram
Switch 7	phase_C_state	binary input and output	Switch is located mid-circuit and is not represented in diagram
Switch 8	phase_A_state	binary input and output	Switch is located mid-circuit and is not represented in diagram
Switch 8	phase_B_state	binary input and output	Switch is located mid-circuit and is not represented in diagram
Switch 8	phase_C_state	binary input and output	Switch is located mid-circuit and is not represented in diagram
Switch 9	phase_A_state	binary input and output	Switch is located mid-circuit and is not represented in diagram
Switch 9	phase_B_state	binary input and output	Switch is located mid-circuit and is not represented in diagram
Switch 9	phase_C_state	binary input and output	Switch is located mid-circuit and is not represented in diagram

Switch 10	phase_A_state	binary input and output	Switch is located mid-circuit and is not represented in diagram
Switch 10	phase_B_state	binary input and output	Switch is located mid-circuit and is not represented in diagram
Switch 10	phase_C_state	binary input and output	Switch is located mid-circuit and is not represented in diagram
Switch 11	phase_A_state	binary input and output	Switch is located mid-circuit and is not represented in diagram
Switch 11	phase_B_state	binary input and output	Switch is located mid-circuit and is not represented in diagram
Switch 11	phase_C_state	binary input and output	Switch is located mid-circuit and is not represented in diagram
Switch 12	phase_A_state	binary input and output	Switch is located mid-circuit and is not represented in diagram
Switch 12	phase_B_state	binary input and output	Switch is located mid-circuit and is not represented in diagram
Switch 12	phase_C_state	binary input and output	Switch is located mid-circuit and is not represented in diagram

Appendix B – Example DNP3 JSON device profile


```
{
  "local_addr": 2,
  "remote_addr": 1,
  "allowUnsol": false,
  "device": {
  "vendor": "SEL",
    "model": "SEL3530",
    "hardware_ver": 1193240101,
    "software_ver": SEL-3530-R145-V0-Z000001-D20190830,
  "functions": [ 1, 2, 13, 14, 20, 21 ],
  "points": {
    "analog_inputs": [
      { "point": 0, "class": 1, "name": "voltage_A_mag", "input_location": "database"},
      { "point": 1, "class": 1, "name": "voltage_B_mag", "input_location": "database"},
      { "point": 2, "class": 1, "name": "voltage_C_mag", "input_location": "database"},
      { "point": 3, "class": 1, "name": "current_A_mag", "input_location": "database"},
      { "point": 4, "class": 4, "name": "current_B_mag", "input_location": "database"},
      { "point": 5, "class": 1, “name": "current_C_mag", "input_location": "database" },
      { "point": 6, "class": 1, "name": "power_out_real", "input_location": "database"},
      { "point": 7, "class": 1, "name": "power_out_imag", "input_location": "database" },
      { "point": 8, "class": 1, "name": "power_out_A_real", "input_location": "database"},
      { "point": 9, "class": 1, "name": "power_out_B_real", "input_location": "database"},
      { "point": 10, "class": 1, "name": "power_out_C_real", "input_location": "database" },
      { "point": 11, "class": 1, "name": "power_out_A_imag", "input_location": "database"},
      { "point": 12, “class": 1, "name": "power_out_B_imag", "input_location": "database"},
      { "point": 13, "class": 1, "name": "power_out_C_imag", "input_location": "database"},
      { "point": 14, "class": 1, "name": "voltage_A_mag", "input_location": "database"},
      { "point": 15, "class": 1, "name": "voltage_B_mag", "input_location": "database"},
      { "point": 16, "class": 1, "name": "voltage_C_mag", "input_location": "database"},
      { "point": 17, "class": 1, "name": "voltage_A_mag", "input_location": "database"},
      { "point": 18, “class": 1, "name": "voltage_B_mag", "input_location": "database"},
      { "point": 19, "class": 1, "name": "voltage_C_mag", "input_location": "database"},
      { "point": 20, "class": 1, "name": "voltage_A_mag", "input_location": "database"},
      { "point": 21, "class": 1, "name": "voltage_B_mag", "input_location": "database"},
      { "point": 22, "class": 1, "name": "voltage_C_mag", "input_location": "database"},
      { "point": 23, "class": 1, "name": "voltage_A_mag", "input_location": "database"},
      { "point": 24, "class": 1, "name": "voltage_B_mag", "input_location": "database"},
      { "point": 25, "class": 1, "name": "voltage_C_mag", "input_location": "database"},
      { "point": 26, "class": 1, "name": "tap_pos_A", "input_location": "database"},
      { "point": 27, "class": 1, "name": "tap_pos_B", "input_location": "database"},
      { "point": 28, "class": 1, "name": "tap_pos_C", "input_location": "database"},
      { "point": 29, "class": 1, "name": "tap_pos_A", "input_location": "database"},
      { "point": 30, "class": 1, “name": "tap_pos_B", "input_location": "database"},
      { "point": 31, "class": 1, "name": "tap_pos_C", "input_location": "database"}
    ],
    "analog_outputs": [
      { "point": 26, "class": 1, "name": "tap_pos_A", "input_location": "database"},
      { "point": 27, "class": 1, "name": "tap_pos_B", "input_location": "database"},
```

```
  { "point": 28, "class": 1, "name": "tap_pos_C", "input_location": "database"},
  { "point": 29, "class": 1, "name": "tap_pos_A", "input_location": "database"},
  { "point": 30, "class": 1, "name": "tap_pos_B", "input_location": "database"},
  { "point": 31, "class": 1, "name": "tap_pos_C", "input_location": "database" }
],
"binary_inputs": [
  { "point": 0, "class": 1, "name": "phase_A_state", "input_location": "database"},
  { "point": 1, "class": 1, "name": "phase_B_state", "input_location": "database"},
  { "point": 2, “class": 1, "name": "phase_C_state", "input_location": "database" },
  { "point": 3, "class": 1, "name": "switchA", "input_location": "database"},
  { "point": 4, "class": 1, "name": "switchB", "input_location": "database"},
  { "point": 5, "class": 1, "name": "switchC", "input_location": "database"},
  { "point": 6, "class": 1, "name": "switchA", "input_location": "database"},
  { "point": 7, "class": 1, "name": "switchB", "input_location": "database"},
  { "point": 8, "class": 1, "name": "switchC", "input_location": "database"},
  { "point": 9, "class": 1, "name": "switchA", "input_location": "database"},
  { "point": 10, "class": 1, "name": "switchB", "input_location": "database"},
  { "point": 11, "class": 1, "name": "switchC", "input_location": "database"},
  { "point": 12, "class": 1, "name": "switchA", "input_location": "database"},
  { "point": 13, "class": 1, "name": "switchB", "input_location": "database"},
  { "point": 14, "class": 1, "name": "switchC", "input_location": "database"},
  { "point": 15, "class": 1, "name": "phase_A_state", "input_location": "database"},
  { "point": 16, "class": 1, "name": "phase_B_state", "input_location": "database"},
  { "point": 17, "class": 1, "name": "phase_C_state", "input_location": "database"},
  { "point": 18, "class": 1, "name": "phase_A_state", "input_location": "database"},
  { "point": 19, "class": 1, "name": "phase_B_state", "input_location": "database"},
  { "point": 20, "class": 1, "name": "phase_C_state", "input_location": "database"},
  { "point": 21, "class": 1, "name": "phase_A_state", "input_location": "database"},
  { "point": 22, "class": 1, "name": "phase_B_state", "input_location": "database"},
  { "point": 23, "class": 1, "name": "phase_C_state", "input_location": "database"},
  { "point": 24, "class": 1, "name": "phase_A_state", "input_location": "database"},
  { "point": 25, "class": 1, "name": "phase_B_state", "input_location": "database"},
  { "point": 26, "class": 1, "name": "phase_C_state", "input_location": "database"},
  { "point": 27, "class": 1, "name": "phase_A_state", "input_location": "database"},
  { "point": 28, "class": 1, "name": "phase_B_state", "input_location": "database"},
  { "point": 29, "class": 1, "name": "phase_C_state", "input_location": "database"},
  { "point": 30, "class": 1, “name": "phase_A_state", "input_location": "database" },
  { "point": 31, "class": 1, "name": "phase_B_state", "input_location": "database"},
  { "point": 32, "class": 1, "name": "phase_C_state", "input_location": "database"},
  { "point": 33, "class": 1, "name": "phase_A_state", "input_location": "database"},
  { "point": 34, "class": 1, "name": "phase_B_state", "input_location": "database"},
  { "point": 35, "class": 1, "name": "phase_C_state", "input_location": "database"},
  { "point": 36, "class": 1, "name": "phase_A_state", “input_location": "database"},
  { "point": 37, "class": 1, "name": "phase_B_state", "input_location": "database"},
  { "point": 38, "class": 1, "name": "phase_C_state", "input_location": "database"},
  { "point": 39, "class": 1, "name": "phase_A_state", "input_location": "database"},
  { "point": 40, "class": 1, "name": "phase_B_state", "input_location": "database"},
  { "point": 41, "class": 1, "name": "phase_C_state", "input_location": "database"},
  { "point": 42, "class": 1, "name": "phase_A_state", "input_location": "database"},
  { "point": 43, "class": 1, "name": "phase_B_state", "input_location": "database"},
  { "point": 44, "class": 1, "name": "phase_C_state", "input_location": "database"},
```

```
  { "point": 45, "class": 1, "name": "phase_A_state", "input_location": "database"},
  { "point": 46, "class": 1, "name": "phase_B_state", "input_location": "database"},
  { "point": 47, "class": 1, "name": "phase_C_state", "input_location": "database"},
  { "point": 48, "class": 1, "name": "phase_A_state", "input_location": "database"},
  { "point": 49, "class": 1, "name": "phase_B_state", "input_location": "database"},
  { "point": 50, "class": 1, "name": "phase_C_state", "input_location": "database" }
],
"binary_outputs": [
  { "point": 0, "class": 1, "name": "phase_A_state", "input_location": "database"},
  { "point": 1, "class": 1, "name": "phase_B_state", "input_location": "database"},
  { "point": 2, "class": 1, "name": "phase_C_state", "input_location": "database"},
  { "point": 3, "class": 1, "name": "switchA", "input_location": "database"},
  { "point": 4, "class": 1, "name": "switchB", "input_location": "database"},
  { "point": 5, "class": 1, "name": "switchC", "input_location": "database"},
  { "point": 6, "class": 1, "name": "switchA", "input_location": "database"},
  { "point": 7, "class": 1, "name": "switchB", "input_location": "database"},
  { "point": 8, "class": 1, "name": "switchC", "input_location": "database"},
  { "point": 9, "class": 1, "name": "switchA", "input_location": "database"},
  { "point": 10, "class": 1, “name": "switchB", "input_location": "database" },
  { "point": 11, "class": 1, "name": "switchC", "input_location": "database"},
  { "point": 12, "class": 1, "name": "switchA", "input_location": "database"},
  { "point": 13, "class": 1, "name": "switchB", "input_location": "database"},
  { "point": 14, "class": 1, "name": "switchC", "input_location": "database"},
  { "point": 15, "class": 1, "name": "phase_A_state", "input_location": "database"},
  { "point": 16, "class": 1, "name": "phase_B_state", "input_location": "database"},
  { "point": 17, "class": 1, "name": "phase_C_state", "input_location": "database"},
  { "point": 18, "class": 1, "name": "phase_A_state", "input_location": "database"},
  { "point": 19, "class": 1, "name": "phase_B_state", "input_location": "database"},
  { "point": 20, "class": 1, "name": "phase_C_state", "input_location": "database"},
  { "point": 21, "class": 1, "name": "phase_A_state", "input_location": "database"},
  { “point": 22, "class": 1, "name": "phase_B_state", "input_location": "database" },
  { "point": 23, "class": 1, "name": "phase_C_state", "input_location": "database"},
  { "point": 24, "class": 1, "name": "phase_A_state", "input_location": "database"},
  { "point": 25, "class": 1, "name": "phase_B_state", "input_location": "database"},
  { "point": 26, "class": 1, "name": "phase_C_state", "input_location": "database"},
  { "point": 27, "class": 1, "name": "phase_A_state", "input_location": "database"},
  { "point": 28, "class": 1, "name": "phase_B_state", "input_location": "database"},
  { "point": 29, "class": 1, "name": "phase_C_state", "input_location": "database"},
  { "point": 30, "class": 1, "name": "phase_A_state", "input_location": "database"},
  { "point": 31, "class": 1, "name": "phase_B_state", "input_location": "database"},
  { "point": 32, "class": 1, "name": "phase_C_state", "input_location": "database"},
  { "point": 33, "class": 1, "name": "phase_A_state", “input_location": "database" },
  { "point": 34, "class": 1, "name": "phase_B_state", "input_location": "database"},
  { "point": 35, "class": 1, "name": "phase_C_state", "input_location": "database"},
  { "point": 36, "class": 1, "name": "phase_A_state", "input_location": "database"},
  { "point": 37, "class": 1, "name": "phase_B_state", "input_location": "database"},
  { "point": 38, "class": 1, "name": "phase_C_state", "input_location": "database"},
  { "point": 39, "class": 1, "name": "phase_A_state", "input_location": "database"},
  { "point": 40, "class": 1, "name": "phase_B_state", "input_location": "database"},
  { "point": 41, "class": 1, "name": "phase_C_state", "input_location": "database"},
  { "point": 42, "class": 1, "name": "phase_A_state", "input_location": "database"},
```

```
      { "point": 43, "class": 1, "name": "phase_B_state", "input_location": "database" },
      { "point": 44, "class": 1, "name": "phase_C_state", "input_location": "database" },
      { "point": 45, "class": 1, "name": "phase_A_state", "input_location": "database" },
      { "point": 46, "class": 1, "name": "phase_B_state", “input_location": "database" },
      { "point": 47, "class": 1, "name": "phase_C_state", "input_location": "database" },
      { "point": 48, "class": 1, "name": "phase_A_state", "input_location": "database" },
      { "point": 49, "class": 1, "name": "phase_B_state", "input_location": "database" },
      { "point": 50, "class": 1, "name": "phase_C_state", "input_location": "database" }
    ]
    }
  }
}
```

PNNL-30387

Pacific Northwest National Laboratory

902 Battelle Boulevard
P.O. Box 999
Richland, WA 99354
1-888-375-PNNL (7665)

www.pnnl.gov