A Real-Time Testbed Environment for Cyber-Physical Security on the Power Grid

Georgia Koutsandria
Dept. of Computer Science
Sapienza University of Rome
Rome, Italy
koutsandria@di.uniroma1.it

Reinhard Gentz
Dept. of Electrical and Computer Engineering
Arizona State University
Tempe, AZ, USA
rgentz@asu.edu

Mahdi Jamei
Dept. of Electrical and Computer Engineering
Arizona State University
Tempe, AZ, USA
mjamei@asu.edu

Anna Scaglione
Dept. of Electrical and Computer Engineering
Arizona State University
Tempe, AZ, USA
Anna.Scaglione@asu.edu

Sean Peisert
Lawrence Berkeley National Laboratory
Berkeley, CA, USA
Dept. of Computer Science
University of California Davis
Davis, CA, USA
sppeisert@lbl.gov

Chuck McParland
Lawrence Berkeley National Laboratory
Berkeley, CA, USA
cpmcparland@lbl.gov

ABSTRACT

The trustworthiness and security of cyber-physical systems (CPSs), such as the power grid, are of paramount importance to ensure their safe operation, performance, and economic efficiency. The aim of many cyber-physical security techniques, such as network intrusion detection systems (NIDSs) for CPSs, is to ensure continuous reliable operation even in exposed network environments. But the validation of such methods goes well beyond standard network analysis, since meaningful tests must also integrate realistic understanding of the physical systems behavior and response to the network activity. Our goal in this paper is to showcase an example of a testbed environment that can support such validation. In it, real network traffic, emulating and industrial control network, interacts with simulated physical models in real-time, extending and leveraging "hardware-in-the-loop" and "cyber-in-the-loop" capabilities. The testbed is a bridge between theory and practice and offers a number of features, including network communications, data management, as well as the virtualization of cyber-physical state analytics performed by the NIDS. The traffic is captured by real network taps and is forwarded to a real data management environment, receiving also the data reports from the simulated industrial control environment. To illustrate the capabilities of our testbed we show how the data are cross-checked by a "physics aware" NIDS, identifying network traffic that does not comply with its cyber-physical security rules.

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems; C.2 [**Computer Communication Networks**]: General—*Security and protection*

Keywords

Testbed, power grid, cyber-physical systems, intrusion detection systems, cyber-physical security.

1. INTRODUCTION

Electric power systems (EPSs) have been early adopters of network technologies for their operations, making them one of the early cases of wide area cyber-physical systems (CPSs). The term "smart grid" in EPS refers to an umbrella of activities aimed at widely expanding the reach of grid monitoring and automation. Among the many goals of the smart grid are improved reliability, efficiency, and sustainability of existing EPS. This will in turn enable a secure, reliable and economic power supply closely coupled with a fast, efficient, and dependable communication infrastructure. However, despite the growing trend of employing CPSs in modern infrastructure, such as the power grid, there are many open questions as to how best to perform experimental security studies on them.

One way of doing security monitoring of network-connected physical devices, such as embedded controllers, might be to monitor the exchanged trustworthy physical data in order to identify outliers in the network traffic using a network intrusion detection system (NIDS) to monitor the packets exchanged. However, in some cases, even if NIDSs are placed in cyber-physical systems, intruders could easily modify the data exchanged without being detected. For that reason, a NIDS might ideally combine knowledge about both cyber and physical parameters. Then, it should be possible to cross-check the information included in the network traffic and used by NIDSs to characterize it as normal or abnormal, with real physical values as they are generated by the field devices. By providing cross-validation, data should theoretically be harder to tamper with in a way that would fool a security monitoring system. In other words, we believe that when developing a testbed environment for cyber-physical systems, one should consider physical and network data history, in the data historians to which network packet flows are sometimes reported.

Then, a NIDS should be able to compare the data in the network traffic with the real physical data in order to accurately conclude about outliers, and, even, be able to retrieve past data. On the other hand, typically, operators in control rooms continuously monitor the values of the physical system in order to identify potential issues which can lead to failures or disturbances, and, based on their observations, take necessary actions for restoring the normal state of the physical system. We believe, that the management domains of cyber and physical security should be merged and operators in control rooms should be aware of the overall cyber-physical state of the system, receiving information in the network traffic to identify possible attacks and launch countermeasures based on the evaluation of the consistency between the cyber and physical events.

However, testing the efficiency of security mechanisms for this new integrated environment, such as intrusion detection methods for cyber-physical systems, is challenging and can often be inconclusive [17]. It is made even harder given that physical effects of network-connected systems are not part of the standard way of analyzing computer security algorithms. In fact, when attempting to draw any valid conclusions at all about security mechanisms, particularly mechanisms for cyber-physical systems, there are at least two essential ingredients necessary to provide an adequate evaluation of security mechanisms: 1) it is necessary to have an accurate and realistic mathematical model that describes the combined cyber and physical normal behavior of the system [12, 19]; 2) it is necessary for an experimental framework to include both "hardware-in-the-loop" and "cyber-in-the-loop" capabilities through a realistic implementation of the components of the CPSs. In this regard, a testbed must either contain an actual physical device or be able to mimic the composite interactions that emulate the orderly behavior of cyber-physical systems.

1.1 Contributions

In this paper, we present a testbed environment for CPSs that focuses on the combination of existing hardware and software components for testing a real-time NIDS. Specifically, our testbed enables the "re-creation" of common architectures of CPSs in order to conduct a series of experiments and test the efficiency of NIDS against cyber-physical attacks or anomalies. We note that details about the implementation and conceptual ideas of the NIDS used in this work, called "HC-NIDS," are presented elsewhere [11, 22, 18], and are beyond the scope of the work presented in this paper. We occasionally refer to this specific NIDS, because, based on the fact that disruptive cyber-physical attack-based experiments on simulation/emulation testbeds can lead to non-realistic results, it is the component responsible for detecting such types of attacks. In other words, our testbed environment is the instrument for leveraging emulation, in some cases, to replicate, in real-time, the realistic cyber-physical behavior of a CPS. In doing so, we were able to test NIDS rules instead having to use real PLCs and actual, connected physical devices in each case. Using physical devices might be prohibitively expensive and might even be destructive to physical environments.

We designed our testbed environment to test an intrusion detection systems for CPSs that we developed by leveraging the capabilities of the Matlab/Simulink environment in order to establish communication over industrial control protocols, e.g., Modbus, with real programmable logic controllers (PLCs). The Matlab/Simulink environment is a well known tool typically used for simulating realistic CPSs. However, there are currently no available features for establishing real network connection with physical devices under standard industrial communication protocols. Therefore, the implementation of this missing feature constitutes one of the contributions of our work. We imagine that other types of safety and security-related experiments could be conducted using our testbed design as well.

In addition, we make use of a data historian in order to report both network traffic "sniffed" by network taps, and physical data as generated by the controllers. We illustrate the capabilities of our testbed using a set of intrusion detection rules that combine both the knowledge of the physical laws and limits of the system under monitoring, and the packets exchanged between the various controllers.

In real-time, we apply an intrusion detection system to the network traffic, and, in cases where it is required by the NIDS, we cross-check the observable traffic with the physical data reported to the data historian by the controllers. In this paper, we focus on the validation framework and extension of the emulation environment to include network taps, data management capabilities, and visualization, whereas the previously published work [11, 22, 18] discusses the derivation of effective NIDS rules. Finally, we show how our system could provide cyber-physical analytics to the operators that merge the view of the physical and cyber-domains and indicate anomalies of the CPS state. Through our implementation, it is simply made clear to the operator what is the current state of the cyber-physical system by visualizing necessary, but adequate, information related to the systems' state and components, without the need of "decrypting" complex representations of the system under monitoring.

The remainder of the paper is organized as follows. Section 2 provides a review of related work. Section 3 introduces the architecture of our testbed environment and presents a description of its capabilities. Section 4 demonstrates the evaluation of our testbed through a number of experimental scenarios. Finally, conclusions are provided in Section 5.

2. RELATED WORK

A variety of testbed environments have been developed in universities, industry, and National Labs to explore CPS security, and evaluate a diverse range of security mechanisms and solutions. In order to develop a testbed for cyber-physical systems, modeling of the physical system and its behavior is a fundamental step for any such solution. The Matlab/Simulink environment is a well known set of software for modelling and analyzing real-time dynamic systems, such as power systems, that supports a number of low-level communication protocols, including TCP/IP. Towards this direction, Martins, et al. [15], focused on the implementation of a Matlab/Simulink framework that simulates the physical process and the control mechanism implemented in PLCs.

However, to provide a complete framework for testing security mechanisms of CPSs, both sides of a cyber-physical system must be considered, and empirical evaluation must be performed to validate theory. Mallouhi, et al. [14], developed TASSCS, a testbed for analyzing security on SCADA systems, that combines the OPNET network simulator and the PowerWorld software to study and evaluate existing security mechanisms and solutions. Their approach is a combination of simulation, emulation, and "hardware-in-the-loop," however, the work they describe only supports communication under the Modbus protocol, and the components of their testbed consist of industrial software that are not open-source, specifically designed to work along with their developed Autonomic Software Protection System (ASPS) techniques. In our case, even though our implementation is based on the Modbus protocol for reasons that are described in later sections, we want to point out that our approach is more general and the choice of the communication protocol is not restricted to the Modbus protocol.

A number of various testbed environments specifically target SCADA systems. Giani, et al. [7], primarily describe the development of a testbed for SCADA systems as they envisioned it at the time the paper was published. The paper gives a good general idea about what should be the architecture of a testbed environment, and which should be its components when considering three distinct deployment architectures: 1) single simulation-based; 2) federated simulation-based; 3) emulation and implementation-based. In fact, the last of the proposed architectures presents similarities, at an abstract level, with the way we envisioned and implemented our testbed, by using industrial physical devices, etc..However, the specific paper is more a description of a work in progress, as well as future work, of the first mentioned architecture. Davis, et al. [5], discussed the development of a fully simulated testbed for cyber security of SCADA systems. However, even though simulating each level of a testbed environment makes the idea of its development easily adaptable, depending on the requirements of the physical system, we believe that by including commercial hardware, and by blending it with existing software solutions, we can validate and test potential security solutions more accurately and realistically.

Adhikari, et al. [1], presented a cyber-physical testbed for the development of an IDS for power systems, that was tested under different type of scenarios, such as power system faults, power system contingencies, and power systems attacks. In their work, they made use of the Snort IDS in order to detect suspicious network activities. In our work, we integrated with the data historian, a novel hybrid control NIDS, named HC-NIDS [11, 22, 18]. More specifically, HC-NIDS cross-checks network traffic exchanged between the various controllers and physical data generated by the physical system, and includes both signature-based and anomaly-based rules by taking physics of the system into account.

Hahn, et al. [9], introduced PowerCyber, a cyber-physical security testbed that consists of control, communication, and physical system components. This work presents some similarities with our work, in the sense of which parts are included in our testbed environment. However, there are some conceptual differences between our testbed and PowerCyber; 1) we make use of a complete NIDS that implements intrusion detection rules by considering both the information from physical system operation and network traffic. Hahn, et al., evaluate attacks based on physical quantities of the system, such as voltage and rotor angle stability; 2) our testbed aims to provide to the operators simple but important information and adequate about the state of system under a wide scope, whereas the Hahn, et al. approach is limited to information reported by field devices.

Hemingway, et al. [10], developed a multimodel-based integration platform to model a complex cyber-physical system in a distributed manner. Their work mainly focuses on how to integrate different simulation engines with semantics that are required to simulate cyber-physical subsystems under a unified framework, called "High Level Architecture (HLA)" and seeks to manage the interaction of these heterogeneous domains. They further extend their work by introducing a novel framework [20] that benefits from the defined standards in HLA for data exchange and time management between different simulation participants, and also uses "Functional Mock-up Interface (FMI)" standards to define the specification and "Application Programming Interface (API)" for simulation components. However, the CPS modeling is entirely addressed in the simulation environment and integration of the "hardware-in-the-loop" and its consequent challenges are not considered.

Fritzson, et al. [6], introduced "OpenModelica", an environment for distributed simulation of complex systems that has been developed using the "Modelica" language. This object-oriented language is designed to be able to provide causal, and multi-domain modeling for different dynamic engineering systems. The aim of the "OpenModelica" is to extend the language, provide an efficient interactive computational environment for it, and to add on top of that visualization and ease-of-use capabilities.

In an effort to implement a co-simulation idea, specifically for the smart power grids as a case of CPSs, Buscher, et al. [2], provided an OpenSource framework, the so-called "mosaik", which enables the functional coupling of different software simulators, emulators, and hardware setups. In fact, mosaik's main goal is to make use of existing simulators and other tools in a common context which means letting each component to execute processes with its own individual event loop and just controlling the synchronization of participants and managing the exchange of data among them. Tan, et. al. [27], developed the "Smart-grid Common Open Research Emulator (SCORE)", that emulates both the power and communication network, and facilitates testing and validating of new ideas in the smart grid. It is intended to fill the gap between simulators and real testbeds, and is designed in a portable and scalable manner. However, they mainly focus on developping the environment of CPSs as a general concept without a security-oriented motivation. Having the security purposes in mind, our work enables the testbed to experiment different attack scenarios and visually validate the efficiency of defense mechanisms.

Siaterlis, et al. [25, 24], presented EPIC, a scientific tool that re-creates the cyber part of networked critical infrastructures (NCIs) based on Emulab and a variety of software simulators for the physical part. EPIC aims on addressing common testbed requirements, such as repeatability, safe execution, fidelity, and measurement accuracy, by running security experiments on multiple heterogeneous NICs. The key difference between EPIC and our testbed environment, is the way in which we make use of the data management in order to cross-check the network traffic and the physical data generated by the physical system, in addition to the fact that the initial motivation behind our approach was to develop a testbed environment that researchers/testers could easily implement even in small research labs by combining well-known and easily accessible hardware and software tools.

Another worth-mentioning work is the one presented by Reddi, et al. [23], that focused on the development of a power system testbed that integrates various hardware to provide modeling, monitoring, and control of power systems in real-time. Even though their work indicates a number of similarities with our work by leveraging common industrial hardware, and a data management system, such as the OSIsoft PI Server system [21], one of the main differences is that they use real time digital simulators (RTDSs). We note that it is far easier and cost effective to emulate real networking hardware and micro-controllers than it is to emulate physical systems, since RTDS are very expensive and allow to test a very limited set of configurations.

In the same conceptual direction, Chen, et al. [3], developed a real-time testbed for CPSs that includes a number of elements, including RTDSs, and the OPNET network simulator. A significant difference between this work and our testbed environment, is that in our case, we *emulate the communication networks* in order to generate real network traffic, instead of simulating them. Furthermore, because the physical system is simulated, in our setting we can accelerate the physical world, and cover more network scenarios. Our approach includes realistic physical models, implemented using the Simulink simulation environment, real embedded controllers, emulated communication networks that follow commonly used industrial protocols in order to generate real traffic, data management, visualization for reporting to the operators cyber-physical analytics, a network tap to sniff and report the network traffic, and a novel NIDS that combines network and physical knowledge to make security decisions and, even though it is out of scope of this work, plays a main role on how the data management handles the incoming network traffic.

3. ARCHITECTURE OF THE TESTBED

In this section, we provide details about the real-time testbed environment that we developed, using an integrated framework of tools for process management and cyber-physical security, and discuss about its capabilities. The testbed consists of a physical system modeled in the Simulink environment, and real and simulated programmable logic controllers (PLCs) that generate real network traffic and exchange packets based on industrial control protocols, in our case the Modbus protocol [26]. In addition, our testbed includes a network tap to capture network traffic exchanged between the various controllers, and a data management for data processing and storing. On top of the underlying testbed deployment, we implemented a NIDS interface that processes streaming data in the historian, continuously monitors the network traffic, applies a set of cyber-physical security policies, in the form of checks, to decide about abnormalities or attacks, and visualizes the results of the checks. Finally, cyber-physical analytics related to the state of the system, including physical devices and communication links among them, are displayed to the network operators in a simple and descriptive manner.

Taking into consideration the complexity of setting up our testbed, it can be configured and monitored from two machines running a different operating system because of limitations on the required operating system by some software elements included in our testbed environment. However, this fact could be easily overcome by using virtual machines that run different operating systems instead of desktop machines. Once the required software and hardware is deployed, there is a specific hierarchy on the execution turn of each element that should be followed in order to achieve synchronization among them. The synchronization is trivial following a common logic, e.g., we should first initiate the execution of the physical system in order to generate data and be able to save them in the data historian.

We also note that our testbed environment supports repeatability in order to obtain the same or statistically similar results. A researcher/tester could easily reproduce a previously tested experiment since the initial state and set up of the controlled environment is fixed at the start of each simulation and is a user-defined choice. In addition, as previously mentioned, the network traffic exchanged between the various controllers is saved in the data management, which makes it easy to the researcher/tester to directly apply the NIDS rules to the saved data, without re-creating a physical environment, and obtain statistically similar results. In addition, we believe that it is feasible, for several users to simultaneously perform experiments because it is possible to run several instances of the same software tool, or even parallel simulations on a software, such as the Matlab/Simulink environment. However, since we did not investigate this capability, it would increase the complexity of the overall set up because it would require partition of the data historian in order to differentiate the network traffic related to each experiment. The general architecture of our testbed environment, which is made up of a number of distinct elements, including the NIDS, a fully description of which is out of scope of this work, is shown in Figure 1.

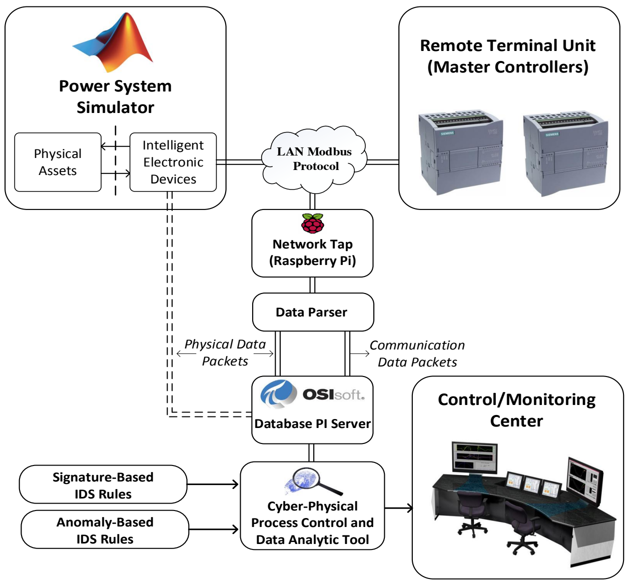
Power System Simulator
Physical Assets
Intelligent Electronic Devices
LAN Modbus Protocol
Remote Terminal Unit (Master Controllers)
Network Tap (Raspberry Pi)
Data Parser
Physical Data Packets
Communication Data Packets
OSIsoft.
Database PI Server
Control/Monitoring Center
Signature-Based IDS Rules
Anomaly-Based IDS Rules
Cyber-Physical Process Control and Data Analytic Tool

Figure 1: General Architecture of the Testbed.

3.1 Physical System

In the real world, power systems obey physical laws that are completely known, such as Kirchhoff's and Ohm's laws.

The voltage and currents power carriers are narrowband signals centered around 50 or 60Hz. The steady state behavior can be easily simulated through algebraic equations and the transient behavior is characterized by ordinary differential equations (ODEs) [4].

The Matlab/Simulink environment provides a number of libraries that assist in representing and simulating the mathematical model which characterizes the behavior of physical sub-systems on the grid, such as transformers, capacitor banks, electric motors, etc.. Graphical block diagrams that simulate the physical laws can be interconnected to simulate the behavior of dynamic systems that have several components. For that reason, the Matlab/Simulink environment is widely used for modeling dynamic systems. There are two very useful features of Simulink that we leveraged in our testbed: 1) its ability to interact with actual micro-controllers and 2) simulating the behavior of hybrid systems, thereby reproducing the interaction between the dynamic system and simulated micro-controllers that are placed on the model and run specific rules.

In addition, it is important to note that the simulated micro-controllers can interact with real micro-controllers and send actual network packets out, a key feature that has made our testbed possible. Even though it is possible to develop algorithms that run on micro-controllers supporting analog and digital connectivity, and serial communications in order to control physical systems, we chose to make use of actual PLCs that support industrial control protocols and we could generate real network traffic. Finally, even though, the type and size of the physical system were not, at least initially, one of our mainly concerns, we note that the Matlab/Simulink environment could be used to model a wide range of physical systems, including large scale electrical grids such as IEEE 14-bus or 30-bus systems.

Specifically, in our work, we developed a Simulink model for a physical system controlled and protected by a real, master programmable logic controller either directly, or by interaction via real Modbus communications with one or more simulated field devices placed on the Simulink model. This architecture is typical in Industrial Control and reflects the hierarchy of Supervisory Control and Data Acquisition (SCADA) systems in which Master Controllers (Remote Terminal Units (RTUs)) interact with several field devices (Intelligent Electronic Devices (IEDs)) to carry out one or more protective missions. Sensor and detector outputs are indicate analog, (e.g., current and voltage magnitude or phase) or digital measurements, e.g., state of a switch. These values are given as input to the next level, which is responsible for embedding them in Modbus packets and exchanging them with the appropriate controllers.

In our tests we assumed that this part is performed by the slave controllers (the IEDs or field devices) that were simulated, when considering a master/slave communication model. Moreover, a number of actuator inputs are assigned to the simulation block and connected either to the simulated IED or directly to the master controller's outputs that interact directly with the physical processes in blocks of the Simulink physical system that we envision, and are directly controlled by the Master (e.g., the RTU). We refer to this type of interaction either as analog or digital feedback, based on the nature of the output signal, since these signals indicate the result of the control mechanism implemented in the master controller.

3.2 Control

Based on the master/slave communication model, the master controller implements the protection and control mechanisms related to the physical system implemented through the Simulink environment. For experimental purposes, we implemented an emulation of a master controller in ladder logic using a real PLC. In addition, in order to create a modular version of our testbed environment, we also created a simulation of the control and protection mechanism, that was implemented in `C` programming language. We want to note, that our approach can be applied to a variety of different types of control and protection mechanisms for CPSs, such as power systems, and can be implemented either by emulation or simulation of the PLCs. In the next two subsections, we provide further details about the implementation of both emulated and simulated controllers.

3.2.1 *Emulation using a real PLC*

Our implementation of emulation using a real PLC uses a Siemens SIMATIC S7-1200 PLC that acts as the master controller and performs the protection function and control mechanism of the physical process. This PLC supports both Ethernet and TCP/IP-based communication protocols, such as the Modbus, allowing communication either as a Modbus client (master) or server (slave). We implemented the control mechanism in Ladder logic using the SIMATIC Software Step 7 Basic. Figure 2 shows the general concept of the communication between emulated and simulated PLCs, and the function that we used. The function block that we used is implemented in Ladder logic and is included in the SIEMENS S7-1200 communication library, as well as the communication between the TIA Portal software and the S-function block.

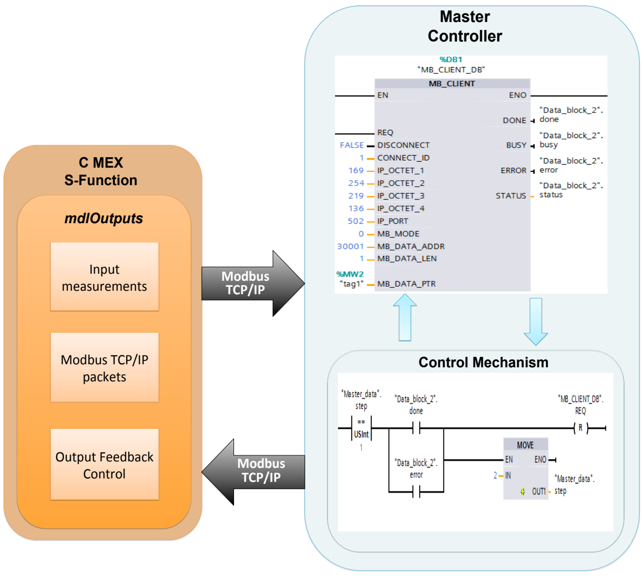

Figure 2: Communication between a simulated slave PLC and an emulated master PLC.

Specifically, we consider a master controller that is responsible for initiating a connection, and exchanges data with the simulated slave controller. The master controller polls the slave controller in order to acquire the value of the input measurements obtained by the related sensors, such as the value of the current that flows on a power transmission line. Then, the protection control algorithm is executed and, based on the result, the master controller sends "write" queries to the slave controller indicating which control action should be performed, such as activating one or more circuit breakers.

3.2.2 *Simulation of a PLC using libmodbus*

The simulation of the master controller was implemented in `C`, and uses the Modbus protocol library `libmodbus` [13] in order to allow communication through the Modbus industrial control protocol. The specific implementation constitutes a replica of the implementation of the protection and control mechanism in ladder logic using the real PLC. The use of a simulated master controller is a more flexible route when no access to real devices is possible, and it also provides substantially greater freedom when choosing the industrial control protocol that will be used to establish communication and exchange data within the network of the controllers. However, even though simulating the software components has always been assumed to be a more efficient way, it could fail to operate in the same way as real devices do under failures of hardware components or computer systems. Figure 3 shows the communication between the simulated PLC written in `C` using the `libmodbus` library and the S-function block.

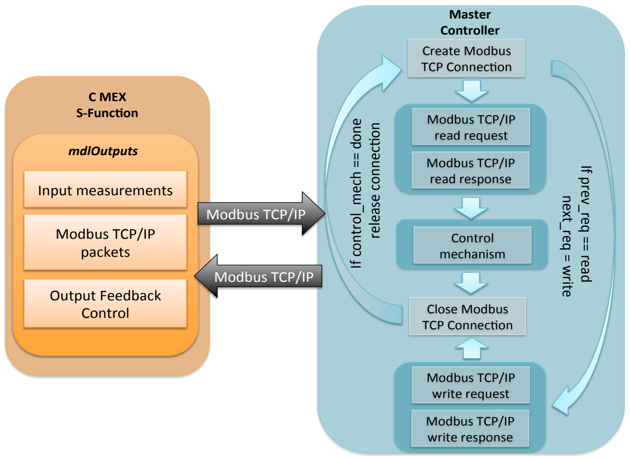
C MEX
S-Function
mdlOutputs
Input measurements
Modbus TCP/IP packets
Output Feedback Control
Modbus TCP/IP
Modbus TCP/IP
Master Controller
Create Modbus TCP Connection
Modbus TCP/IP read request
Modbus TCP/IP read response
Control mechanism
Close Modbus TCP Connection
Modbus TCP/IP write request
Modbus TCP/IP write response
If control_mech == done release connection
If prev_req == read next_req = write

Figure 3: Communication between the Matlab/Simulink environment and a simulated PLC.

Similarly to the procedure described above in the description of the emulation, the master controller establishes a connection with the slave controller and dispatches "read" requests in order to obtain the value of physical measurements. Then, the simulated master controllers perform the protection control algorithm related to the protection of the physical system, and dispatches the appropriate queries indicating the result of the protection mechanism via control actions. Also, in between the two different types of requests, the master controllers releases the connection. Algorithm 1 shows the generalized sequence of the primary functions executed in the `C` code our case study, where we assume that the master controller receives analog values as inputs and returns digital values. However, the `C` code can be customized for different cases based on the requirements of the application, as well as the control mechanism, the code of which is not shown in Algorithm 1 since it is related to the chosen protection and control mechanism.

Algorithm 1 Simulation of a PLC

```
modbus_t $*ctx$
int $rc$
bool sts
char $*modbusClientIP \leftarrow$ CLIENT_IP
int $modbusPort \leftarrow$ PORT
$ctx \leftarrow$ modbus_new_tcp($modbusClientIP, modbusPort$)
if $ctx ==$ NULL then
    fprintf(Unable to allocate libmodbus context)
    $return \leftarrow -1$
end if
if modbus_connect($ctx$) $== -1$ then
    fprintf(Connection failed)
    $sts \leftarrow$ false
end if
$rc \leftarrow$ modbus_read_input_registers
C code of control mechanism
modbus_close($ctx$)
modbus_free($ctx$)
$ctx \leftarrow$ modbus_new_tcp($modbusClientIP, modbusPort$)
if $ctx ==$ NULL then
    fprintf(Unable to allocate libmodbus context)
    $return \leftarrow -1$
end if
if modbus_connect($ctx$) $== -1$ then
    fprintf(Connection failed)
    $sts \leftarrow$ false
end if
$rc \leftarrow$ modbus_write_bits
modbus_close($ctx$)
modbus_free($ctx$)
```

3.3 Communication

The part of the communication between the physical model implemented using the Matlab/Simulink environment and actual controllers constituted one of the challenges that we had to overcome during the development of our testbed. Among the many capabilities that Matlab supports, we decided to use the option of direct communication under a client/server architecture, including interfaces to support TCP and UDP protocols. Our vision was to develop a testbed that could support various industrial control protocols that run over TCP or UDP, such as DNP3 and IEC 61850, by either leveraging existing libraries, an option that is not difficult to apply, or by "formatting" the packets based on the appropriate packet frame format. In this way, a potential user could easily modify our initial implementation to adjust the communication type based on the requirements of the current controllers or preferences among the different industrial control protocols.

Our real-time testbed environment allows us to establish communication between a simulated physical process and a real or a simulated PLC through an Ethernet interface that sends information via the Modbus industrial control protocol. While many cyber-physical processes rely on more advanced options, the choice of Modbus for our experimental validation is dictated by practical convenience: PLCs are inexpensive and a variety of software tools and libraries exist to communicate with them. However, the ideas that we present in this paper are applicable to many of the application layer protocols used in industrial and control systems. We do note however, that different methods would need to be employed if the network communications are encrypted.

Specifically, we leveraged the capabilities of Matlab that supports TCP/IP communication in order to enable communication and transfer data over the Modbus industrial control protocol. We developed an S-function block that was implemented in `C` and which uses the open source Modbus protocol library `libmodbus` in order to simulate the communication behavior of a controller on a linux operating system. The S-function code is compiled using the Matlab mex compiler [16], which also creates a dynamically-loadable executable used by the Simulink model. The same approach could be used to allow communication through other industrial control protocols by including the S-function code for the appropriate protocol libraries instead of the Modbus protocol library.

The S-function block receives as inputs the outputs of the physical process implemented using the Simulink environment, and allows communication with a second PLC. The sensors and detectors included in the dynamic model provide the S-function block with the appropriate physical measurements, which are used in the next level in order to check the consistency of the physical model. Then, the S-function block is responsible for formulating packets that include the input measurements, and dispatches the Modbus packets to the master controller whenever a “read” command query is received. The S-function block also performs the feedback control actions specified in a “write” query, which the master controller sends after the protection and control mechanisms are executed. In addition, the time of each measurement sample is also given as an input to the S-function block in order to process a specific number of samples and calculate an estimate of the specific measured variable within a given time frame. The S-function begins by including and defining the necessary libraries, headers, and variables, including the following:

- `libmodbus` library
- width of analog input/outputs
- width of digital input/output ports
- client’s IP address
- sampling frequency.

Then, the S-function method *mdlInitializeSizes* is executed to set up the characteristics of the S-function block:

- assignment of analog inputs/outputs to specific ports
- assignment of digital inputs/outputs to specific ports
- data types of analog inputs/outputs
- data types of digital inputs/outputs.

The S-function method *mdlInitializeSampleTimes* indicates the sample rates, where in our case the value CONTINUOUS_SAMPLE_TIME is used for continuous sample rate, and the *mdlStart* method is executed only once for initialization purposes. In the next step, the S-function *mdlOutputs* function method is executed and is responsible for acquiring the measurements from the Simulink model, formulating the Modbus packets, and computing the output signals of the S-function block that corresponds to the feedback of the control mechanism. The S-function function method *mdlTerminate* is called at the end of the code to perform actions such as emptying memory. Figure 4 shows the sequence of the functions executed within the S-function block.

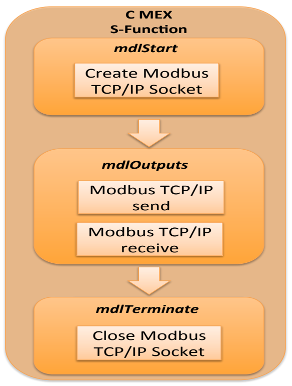

Figure 4: Methods executed in the S-function block.

3.4 Network Tap

The purpose of the network tap is to capture the network traffic exchanged between the controllers that are embedded in the Matlab/Simulink hybrid physical system simulation, and the real controller (PLC) that implements a protection mechanism and interacts with the physical model. We implemented the specific element using a Raspberry Pi and an Ethernet hub where all the physical PLC devices are connected. The choice of the Raspberry Pi was dictated by to factors: 1) low-cost hardware; 2) supports two separate network interfaces that allowed us to split connections among control devices and the data management.

More in detail, in this stage, a Python script using the libcap/pcapy library runs on the Raspberry Pi in order to capture the network traffic. The libcap/pcapy library is an open source tool which can be easily used on several platforms. The captured network traffic is then reported via a second Ethernet card to our data historian and processed so that it can be visualized for the operator. The idea is that the industrial control network and the network aimed at monitoring the traffic should be isolated, reducing potential attack vectors. Also our network tap cannot be used to access or attack the physical devices, as firewall rules do not allow any outward traffic on the Ethernet card doing the packet capture. A possible attack could occur by gaining root access on the Raspberry Pi, however, by closing all incoming ports and discarding any incoming traffic but acknowledgements on the data management interface, this threat could be restricted to a case that requires physical access to the device.

3.5 Data Management

The data management part consists of the OSIsoft [21] PI System, one of the industry-leading data management systems. We chose to use the PI System because it is often already in place in industrial environments, particularly power utilities. Second, the PI System provides a complete framework for data management and archiving. In our setup the PI Server receives data from the network tap and archives it in the form of pre-defined tags. The PI Sever can be either queried online or offline by the NIDS, which implements a set of security policies and is aware of the physical laws and communication rules that designate normal behavior of the cyber-physical system. Specifically, the PI Server performs two actions: 1) stores data from the network tap; 2) retrieves stored data when queried by the NIDS.

One of the challenges arising when considering data historians in CPSs is data volume. A CPS typically includes a large number of sensors that generate data every few minutes; these are the records stored in data historians for processing reasons. The process data historian stores data of fixed size in individual fields, in our case in the form of tags, the amount of which after a certain period and depending on memory capabilities of the various databases could cause memory overflow since the stored data exceed the capacity of the database system. Therefore, conventional data historians integrated to testbed environments that handle a large amount of data need to consider additional solutions to address issues relating to high data volume, e.g., using compression, or, as in our case, periodically deleting old data.

The implementation of the above mentioned capabilities is a combination of `C++` and Python programming languages. Under the phase where we need to read data from the network tap, a Python script is responsible for processing the captured data, and, then, "forward" the processed data to the data historian. As Python does not have a direct support for communication with the PI Server we are using `C++` program as middleware. More specifically, the execution of the steps during this phase for each packet is as follows:

1. Python script accesses the `C++` script, and either establishes connection with the network tap (real time) or recorded files (offline). Timestamps are extracted from original capture time.
2. Network packets are splitted based on the various header fields; only well-formed packets are processed.
3. Filter for Modbus packets (TCP port 502), and classify packets based on their type, e.g., query or response, based on their port (source or destination).
4. Extract the fields corresponding to specific tags:
   (a) plc_source_ip and plc_destination_ip
   (b) plc_transport_identify and plc_protocol_identify
   (c) plc_length_modbus
   (d) plc_unit_identifier
   (e) plc_function_code
5. Based on the type of packet and the command function code, decode the remaining datastream in the form of tags:
   (a) plc_reference_number
   (b) plc_bit_count and plc_byte_count
   (c) plc_payload
   (d) plc_query_or_response
6. In cases where specific fields are not included in the packet frame under processing, assign "-1" to corresponding tags.

As soon as the data processing is completed, the next phase is to store data to the PI Server in the form of tags. The `C++` script is now responsible for executing the following procedure for each field in a packet:

1. Create a socket to connect to the Python script, and connect to PI Server.
2. Receive data and timestamp form Python script in the form of tags, acknowledge each received tag; order of arguments defines which tag to write at each time. Individual acknowlegements for each item ensure delivery and ensure that the socket traffic is not unintentionally combined.

Having followed the above described procedures, the PI Server keeps archives in the form of tags related to the network traffic captured by the network tap. Now, we consider the phase where our NIDS wants to retrieve data stored in the PI Server in order to execute the set of cyber-physical security policies. Based on the fact that the PI Server follows a "pull" architecture, we need to query the server in order to retrieve the stored samples. In our implementation, we chose to query the server every 10ms, which based on our observations constitutes a good compromise between delay and overhead. The NIDS is implemented in Python scripts, which should connect to the `C++` script that is responsible for querying the PI Server and retrieving stored data. Here, we only mention the steps followed by the NIDS in order to gain data, since a description of the security policies in the NIDS is out of the scope of this paper.

1. Python script connects to `C++` script at a given IP port, and sends read Tag cmd to the `C++` script.
2. Once an answer is received, send an acknowledgment, wait for a "req done", and proceed to the next tag.

On the data management side, the `C++` script follows a number of steps in order to connect to the PI Server, and query it for data requested by the NIDS:

1. Open a socket with Python script, and connect to PI Server.
2. Once a Python script request is received, query the PI Server for the specified tag initial timestamp and number of samples that the script requested.
3. Check whether PI Server returned the right amount of samples. In case there are not enough samples available, retry.
4. Check if timestamp is valid.
5. Send values and timestamps to Python script, and "req done" to Python script once the request is completed.

Back to the big data problem, as we archive the complete set of network traffic in our system, we are capturing large quantities of data. While our scenario was small and limited in time we notice that there is the additional need to delete obsolete data from the archive, as otherwise network traffic will rapidly fill the available storage. One might mark data as obsolete after a certain amount of time to allow at least some backlogging and further investigation after an alert from our system. In the case of the PI Server that we used in our testbed, PI Server includes a configuration file where it is possible to globally set the option to delete pi data after a predefined number of GB. However, this is highly dependent on the environment in which the system is deployed, and should be addressed by the consumer.

3.6 Network Intrusion Detection System

The top level of our testbed environment includes a NIDS that implements a "hybrid" set of specification-based and anomaly-based IDS rules by blending common network communication policies with physical laws and limits that designate the expected operation of the physical system. However, the implementation of the hybrid control NIDS, including development of the cyber-physical security rules included in the NIDS, and its performance, is out of scope of this paper and is described further elsewhere [11, 22, 18].

However, we do note that in our testbed deployment, the NIDS was implemented using Python scripts and is related to case studies of specific physical systems that we chose to evaluate its performance. Therefore, we note that the NIDS in our testbed environment can be customized based on the requirements and physical constraints of the physical system, and fully integrated to the overall testbed environment. Based on our observations during the set of experiments that we implemented, we believe that some difficulties may occur in cases where a complete NIDS framework is chosen instead of our version of implementing a set of intrusion detection rules in Python scripts. Network monitoring frameworks can be used to develop a NIDS but typically they make use of the network traffic in a direct way, i.e., by initially capturing the network traffic and processing them to extract necessary fields. In our case, our Python script queries the data historian based on the pi tags and makes direct use of them without further processing. As result, in order to be able to use a network monitoring framework to built a NIDS, it should be modified to accept already processed fields of the network traffic instead of whole network packets.

3.7 Cyber-Physical Visualization & Analytics

The final part of our testbed environment is the visualization of information in a form that is relatively easy to understand while still providing sufficient detail about results of cyber-physical checks performed by the NIDS to be actionable. Hence, our aim was to provide to the operators or administrators of the physical system with a complete picture that includes a description of the cyber-physical state of the system. The visualization part, which is described more in detail later in Section 4, includes information concerning the following physical and cyber aspects:

1. Master controllers, and their IP addresses
2. Slave controllers that correspond to specific physical devices, e.g., power transformers, and the protection schemes protecting them
3. Network communication links between the various controllers
4. Values of physical quantities, e.g., current, voltage, etc., and physical devices, e.g., circuit breakers, as well.
5. Results of the checks performed in the NIDS
6. Statistical information when considering anomaly detection
7. Alerts in cases of abnormalities or attacks.

The capabilities mentioned above provide a small set of features that our visualization software could support. Depending on the physical system, the security policies implemented in the NIDS, and the physical or network values that one chooses to monitor or display, we believe that it can be easily extended in order to include more information. In our case, we implemented a set of features which we believe that could help CPS administrators understand abnormalities or attacks in a descriptive way.

4. TESTBED EXPERIMENTATION

To evaluate our real-time testbed, we focused on the operation of a power transformer on a transmission line that was implemented in the Simulink simulation environment, similarly to that mentioned in the previously published work [11]. In this section, we focus on the cyber-physical analytics that are the result of the HC-NIDS [11], to help provide a visualization of the cyber-physical state for an operator.

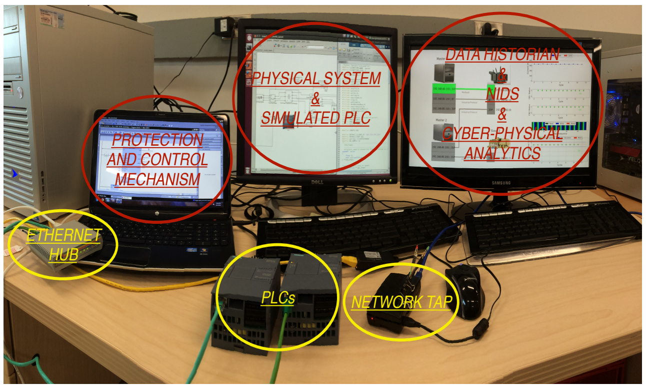

Figure 5: Deployment of our testbed environment.

We note that the purpose of this section is to highlight the way that the cyber-physical analytics are presented to the operators through the visualization of the system's state, and not an evaluation of the performance of the HC-NIDS. Figure 5 shows an example deployment of our testbed environment, including some of the actual hardware and software used in it, such as an ethernet hub, commercial PLCs, a network tap implemented using a Raspberry Pi, the Simulink model of the physical model, and the visualization of the cyber-physical analytics.

In our testbed environment, an operator is able to continuously see the state of the system via the representation of the status as it is reported by the HC-NIDS after the check of a set of intrusion detection rules implemented by the HC-NIDS. This set of intrusion detection rules is the result of the expected operation of the system by combining physical aspects (e.g., value of current, acceptable commands), and cyber aspects (e.g., acceptable IP addresses, protocol ID). Figure 6 shows the visualization of the cyber-physical analytics that our real-time testbed environment makes available to operators. More specifically, our visualization is divided into two parts:

1. Left part: represents the various physical devices under monitoring, including components of the physical system, protection mechanisms as device numbers as reported in the IEEE C37.2 standard, controllers, the network connections between them, and the various IP addresses related to each physical device.
2. Right part: represents values of physical quantities, e.g. transformer's current, status of circuit breakers, and time gap between packets, which are all related to the cyber-physical rules included in the HC-NIDS.

An operator monitoring the state of the system is shown the following: 1) an anomaly or attack was detected; 2) the corresponding rule that triggered the alert; 3) network traffic between which components was characterized as suspicious. In the case shown in Figure 6, the part of the system that is monitoring (highlighted with green color) corresponds to the scenario that we implemented of a power transformer's overcurrent protection, where the green color indicates that the system is under a normal state. In the next subsections, we present a number of experimental cases that correspond to different attack scenarios and how their detection is displayed to the operators.

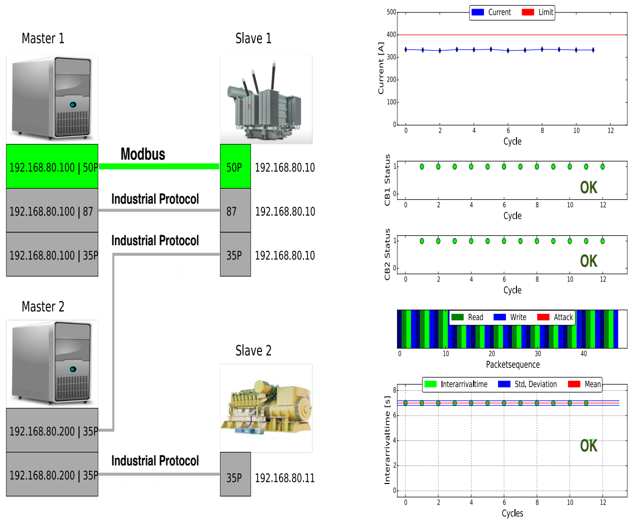

Figure 6: Cyber-physical analytics in a normal state of the physical system under monitoring.

4.1 Packet sequence

In this case, shown in Figures 7–8, we show the network traffic as a packet sequence where different colors represent different data items depending on source and destination devices, function codes, registers, sequence, and value range.

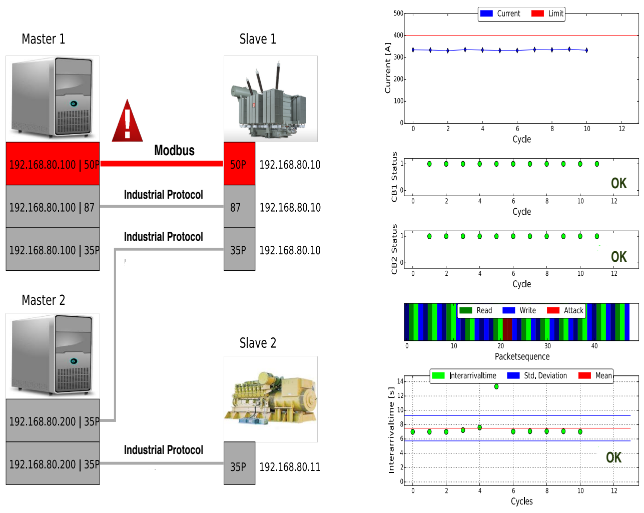

Figure 7: Cyber-physical analytics for an abnormality or attack on packet sequence.

The HC-NIDS identifies as normal traffic packets that appear in a "read query (dark green)-read response (light green)-write query (dark blue)-read query(light blue)" sequence. Packet sequences that deviate from the desired expected packet sequence are considered by the HC-NIDS as anomalies or attacks and are indicated with a red color. Since a query is, typically, followed by a response, in cases of an anomaly or attack we expect to observe one packet that is related to the query marked as red, and another one indicated again with red color about the corresponding response packet.

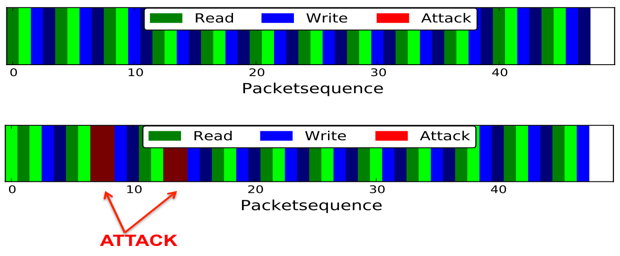

Figure 8: Packet sequence under normal and attacked state.

4.2 Status of circuit breakers

Based on our case study, the operator can monitor the status of the circuit breakers included in our physical system, in addition to the power transformer's current. The right combination on their status indicates whether the protection mechanism is active or not, e.g., whether an overcurrent event was detected.

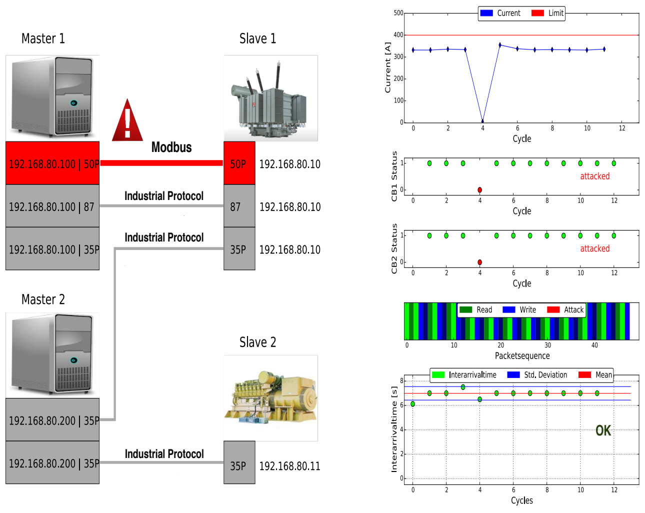

Figure 9: Cyber-physical analytics for an abnormality or attack on the status of circuit breakers.

In cases where an attacker performs a malevolent action, e.g., by changing the status of the circuit breakers, when there is no need to do so based on the protection mechanism, the operator is shown a vizualization about the possible attack or abnormality, and can proceed to further actions, as we observe in Figures 9-10. More specifically, Figure 10 shows that in the case of an over-current event (left part) the protection mechanism was performed correctly by activated the circuit breakers, whereas in the case of a non over-current event (right part) the NIDS considered the activation of the circuit breakers as a possible attack or abnormality. In the last case where a possible attack is identified, the status of the circuit breakers is displayed in red color, whereas green color indicate a normal status.

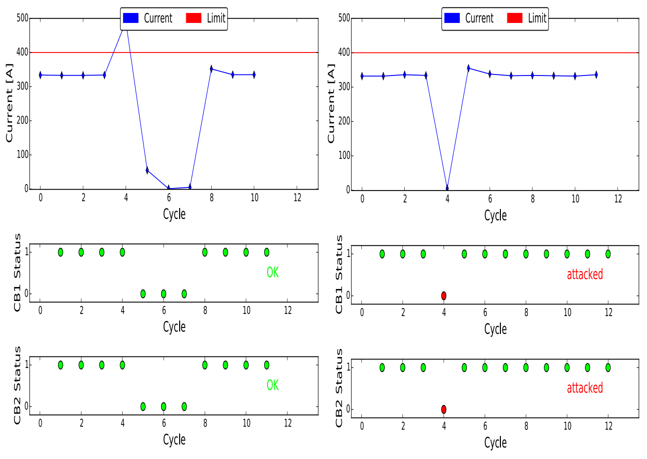

Figure 10: Status of circuit breakers under normal and attacked state.

4.3 Time gap between a cycle of packets

Here, we focus on the time gap between a cycle of packets exchanged, where we define a cycle as the exchange of a "write" and "read" command, including their responses. In this case, network traffic is characterized as attack/anomaly based on statistical properties of arrival time of packets. The implementation of the specific rule in the HC-NIDS constitutes an improvement over the work presented in [11], by including anomaly-based intrusion detection rules. More specifically, we make use of the two-sided CUSUM technique [8] in order to provide accurate time change detection of the time gap between subsequent cycles.

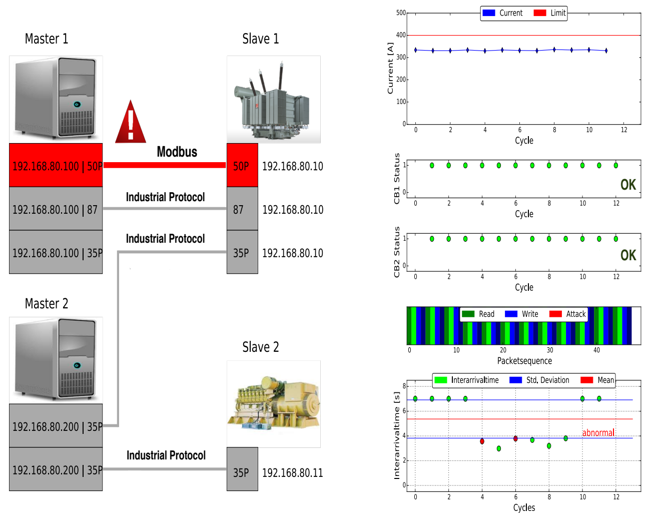

Figure 11: Cyber-physical analytics under an abnormality or attack on the time gap within a cycle.

Figures 11-12 show that time differences between "read" command requests that exceed a pre-defined threshold, indicate an abnormality. The operator has a clear view of statistical information related to packet arrival time, such as the mean and the standard deviation. In cases where an abnormality is detected, the time gap is displayed in red color, rather than green, which corresponds to a normal state.

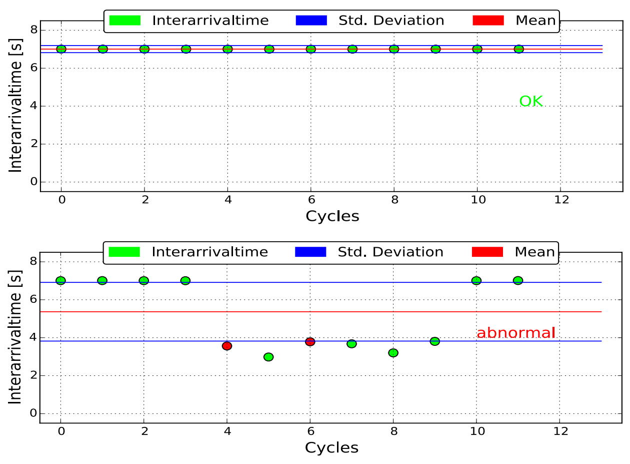

Figure 12: Time gap between a cycle of packets under normal and abnormal state.

5. CONCLUSIONS

Networked CPSs facilitate remote interaction, which makes them more convenient to access, but also makes them vulnerable to a variety of cyber threats. To address this issue, CPSs should support cyber-physical security mechanisms, such as NIDSs, in order to alert control administrators about suspicious activities. Hence, there is a need to develop testbed environments that can be used to evaluate the effectiveness of potential security solutions. We believe that a testbed environment for cyber-physical security of CPSs should provide a realistic blend of hardware, software, and networking components. A realistic testbed environment should include capabilities to analyze the complex relationships that are formed between the physical system, control mechanisms, and network-based security solutions.

In this paper, we discussed our testbed environment for validating a novel NIDS for industrial control processes by monitoring actual network traffic among controllers that perform a number of control actions in a simulated physical environment. More specifically, initially, we leveraged the capabilities of Matlab/Simulink simulation environment in simulating physical processes to create a real communication between the simulated physical system and actual controllers through common industrial communication protocols, such as Modbus. A network tap captures traffic that flows within the network of the simulated controller and the emulated controller that implements protection mechanisms.

In addition, we make use of a widely-applied industrial data management system, the OSIsoft PI System, to store the captured network traffic in the form of predefined tags, as well as data reported by the simulated physical system. Then, we apply a novel NIDS, that retrieves the stored data, cross-checks the data, and evaluates the data for possible anomalies or attacks using both "cyber" and "physical" elements. Finally, our testbed environment includes visualization features for making available to the operators of the infrastructure cyber-physical analytics about the state of the system based on results of the checks implemented in the set of intrusion detection rules.

6. ACKNOWLEDGMENTS

This research was supported in part by the Director, Office of Science, and the Director, Office of Electricity Delivery and Energy Reliability, of the U.S. Department of Energy, under contract DE-AC02-05CH11231. It is also supported in part by the Department of Energy under Award Number DE-OE0000097 and by the National Science Foundation under Grant Number CCF-1018871. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsors of this work.

7. REFERENCES

[1] U. Adhikari, T. H. Morris, and S. Pan. A Cyber-Physical Power System TestBed for Intrusion Detection Systems. In *Proc. of the PES General Meeting | Conference & Exposition*, July 2014.

[2] M. Buscher, A. Claassen, M. Kube, S. Lehnhoff, K. Piech, S. Rohjans, S. Scherfke, C. Steinbrink, J. Velasquez, F. Tempez, et al. Integrated Smart Grid Simulations for Generic Automation Architectures with RT-LAB and Mosaik. In *Proc. of the International Conference on Smart Grid Communications*, pages 194–199, Nov. 2014.

[3] B. Chen, K. L. Butler-Purry, A. Goulart, and D. Kundur. Implementing a Real-Time Cyber-Physical System Test Bed in RTDS and OPNET. In *Proc. of the North American Power Symposium (NAPS)*, Sep. 2014.

[4] E. A. Coddington and N. Levinson. *Theory of Ordinary Differential Equations.* Tata McGraw-Hill Education, 1955.

[5] C. Davis, J. Tate, H. Okhravi, C. Grier, T. Overbye, and D. Nicol. SCADA Cyber Security Testbed Development. In *Proc. of the 38th North American Power Symposium*, pages 483–488, 2006.

[6] P. Fritzson, P. Aronsson, H. Lundvall, K. Nyström, A. Pop, L. Saldamli, and D. Broman. The OpenModelica Modeling, Simulation, and Development Environment. In *Proc. of the 46th Conference on Simulation and Modelling of the Scandinavian Simulation Society*, Oct. 2005.

[7] A. Giani, G. Karsai, T. Roosta, A. Shah, B. Sinopoli, and J. Wiley. A Testbed for Secure and Robust SCADA Systems. *SIGBED Rev.*, 5(2):4:1–4:4, 2008.

[8] P. Granjon. The cusum algorithm a small review. `http://chamilo2.grenet.fr/inp/courses/ENSE3A35EMIAAZ0/document/change_detection.pdf`.

[9] A. Hahn, A. Ashok, S. Sridhar, and M. Govindarasu. Cyber-Physical Security Testbeds: Architecture, Application, and Evaluation for Smart Grid. *Transactions on Smart Grid*, vol. 4:847–855, 2013.

[10] G. Hemingway, H. Neema, H. Nine, J. Sztipanovits, and G. Karsai. Rapid Synthesis of High-level Architecture-based Heterogeneous Simulation: A Model-based Integration Approach. *Simulation*, 88(2):217–232, Feb. 2011.

[11] G. Koutsandria, V. Muthukumar, M. Parvania, S. Peisert, C. McParland, and A. Scaglione. A Hybrid Network IDS for Protective Digital Relays in the Power Transmission Grid. In *Proc. of the International Conference on Smart Grid Communications*, pages 908–913, Nov. 2014.

[12] A. Kusiak. *Computational Intelligence in Design and Manufacturing.* John Wiley & Sons, 2000.

[13] Libmodbus. `http://libmodbus.org`.

[14] M. Mallouhi, Y. Al-Nashif, D. Cox, T. Chadaga, and S. Hariri. A Testbed for Analyzing Security of SCADA Control Systems (TASSCS). In *Proc. of the PES Innovative Smart Grid Technologies*, Jan. 2011.

[15] J. Martins, C. Lima, H. Martínez, and A. Grau. A Matlab/Simulink Framework for PLC Controlled Processes. *Matlab-Modelling, Programming and Simulations*, 2010.

[16] MATLAB. Matlab mex compiler. `http://www.mathworks.com/help/matlab/ref/mex.html`.

[17] J. McHugh. Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Laboratory. *Transactions on Information and System Security*, 3(4):262–294, Nov. 2000.

[18] C. McParland, S. Peisert, and A. Scaglione. Monitoring Security of Networked Control Systems: It's the Physics. *Security & Privacy*, 12(6):32–39, Nov/Dec 2014.

[19] S. B. Morriss. *Automated Manufacturing Systems: Actuators, Controls, Sensors, and Robotics.* Glencoe/McGraw-Hill, 1994.

[20] H. Neema, J. Gohl, Z. Lattmann, J. Sztipanovits, G. Karsai, S. Neema, T. Bapty, J. Batteh, H. Tummescheit, and C. Sureshkumar. Model-Based Integration Platform for FMI Co-Simulation and Heterogeneous Simulations of Cyber-Physical Systems. In *Proc. of the 10th International Modelica Conference*, pages 10–12, 2014.

[21] OSIsoft. `http://www.osisoft.com/`.

[22] M. Parvania, G. Koutsandria, V. Muthukumary, S. Peisert, C. McParland, and A. Scaglione. Hybrid Control Network Intrusion Detection Systems for Automated Power Distribution Systems. In *Proc. of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 774–779, 2014.

[23] R. M. Reddi and A. K. Srivastava. Real Time Test Bed Development for Power System Operation, Control and Cyber Security. In *Proc. of the North American Power Symposium*, pages 1–6, 2010.

[24] C. Siaterlis and B. Genge. Cyber-Physical Testbeds. *Communications of the ACM*, 57(6):64–73, 2014.

[25] C. Siaterlis, B. Genge, and M. Hohenadel. EPIC: A Testbed for Scientifically Rigorous Cyber-Physical Security Experimentation. *Transactions on Emerging Topics in Computing*, 1(2):319–330, 2013.

[26] A. Swales. Open Modbus/TCP Specification. *Schneider Electric*, vol. 29, 1999.

[27] S. Tan, W.-Z. Song, Q. Dong, and L. Tong. Score: Smart-Grid Common Open Research Emulator. In *Proc. of the Third International Conference on Smart Grid Communications*, pages 282–287, Nov. 2012.