Modeling DNP3 Traffic Characteristics of Field Devices in SCADA Systems of the Smart Grid

Huan Yang*, Liang Cheng† and Mooi Choo Chuah‡
Department of Computer Science and Engineering, Lehigh University, Bethlehem, PA 18015
Email: *huy213@lehigh.edu, †cheng@cse.lehigh.edu, ‡chuah@cse.lehigh.edu

***Abstract*—In the generation, transmission, and distribution sectors of the smart grid, intelligence of field devices is realized by programmable logic controllers (PLCs). Many smart-grid subsystems are essentially cyber-physical energy systems (CPES): For instance, the power system process (i.e., the physical part) within a substation is monitored and controlled by a SCADA network with hosts running miscellaneous applications (i.e., the cyber part). To study the interactions between the cyber and physical components of a CPES, several co-simulation platforms have been proposed. However, the network simulators/emulators of these platforms do not include a detailed traffic model that takes into account the impacts of the execution model of PLCs on traffic characteristics. As a result, network traces generated by co-simulation only reveal the impacts of the physical process on the contents of the traffic generated by SCADA hosts, whereas the distinction between PLCs and computing nodes (e.g., a hardened computer running a process visualization application) has been overlooked. To generate realistic network traces using co-simulation for the design and evaluation of applications relying on accurate traffic profiles, it is necessary to establish a traffic model for PLCs. In this work, we propose a parameterized model for PLCs that can be incorporated into existing co-simulation platforms. We focus on the DNP3 subsystem of slave PLCs, which automates the processing of packets from the DNP3 master. To validate our approach, we extract model parameters from both the configuration and network traces of real PLCs. Simulated network traces are generated and compared against those from PLCs. Our evaluation shows that our proposed model captures the essential traffic characteristics of DNP3 slave PLCs, which can be used to extend existing co-simulation platforms and gain further insights into the behaviors of CPES.**

I. Introduction

Supervisory Control and Data Acquisition (SCADA) system is a critical component of the smart grid, supporting information exchange among field devices, local human-machine interfaces (HMI), as well as remote control center servers. Due to stringent requirements on electromagnetic compatibility (EMC), intelligence of field devices in the generation, transmission, and distribution sectors of the smart grid is implemented by programmable logic controllers (PLCs). Another advantage of PLCs is its standardized programming languages [1], such as ladder logic, which make it easy to both debug new PLC programs and port existing ones.

To understand the behaviors of a cyber-physical energy system (CPES) when certain events occur (e.g., transient faults caused by a lightning strike), it is important to model the traffic characteristics of PLCs because they are the major interfaces between its cyber and physical parts: Control and protection tasks typically require data (e.g., status of circuit breakers) and actions (i.e., command execution) of multiple field devices. As the smart grid continues to grow in complexity and interconnectivity, failures of its cyber components, e.g., due to software bugs or system events, can result in significant impacts on its physical processes. Co-simulation (e.g., [2], [3]) allows us to leverage tools developed by various research communities and comprehensively assess such impacts. Realistic datasets generated by co-simulation are extremely valuable because access to real SCADA datasets is strictly controlled by SCADA system operators due to security and privacy concerns. However, existing work on co-simulation mainly focuses on the concrete mechanisms for information exchange and synchronization among domain-specific tools. To enable co-simulation platforms to generate realistic SCADA traffic datasets for the design and evaluation of applications relying on traffic profiles of SCADA hosts, it is important to incorporate traffic characteristics of PLCs and analyze the time it takes for commands/data to get prepared for transmission over the SCADA network because the network-induced delays are comparable in magnitude to the processing delays incurred by a PLC program. For instance, it is reported in [4] that, for a SCADA network consisting of an Ethernet switch and 21 field devices, the maximum network-induced delays is about 200 μs. According to [5], it takes about 1 μs to execute a PLC comparison instruction (e.g., the `EQU` instruction for equality test). PLC programs of reasonable sizes can thus take up to several hundreds of microseconds to execute.

Recently, cyber threats targeting the smart grid, such as malware and denial-of-service (DoS) attacks, have become a major concern since there is a need to interconnect SCADA systems of different power-system sectors. Among existing methods for cyber-threat detection, those based on traffic behavior analysis (e.g., [6]) are well-suited for SCADA networks because they do not depend on the packets payload and thus can work with encrypted SCADA traffic data. In contrast to computers running desktop operating systems, PLCs execute control programs in a significant different manner (see Sec. III). It has been shown that SCADA traffic exhibits characteristics that are significantly different from those of the Internet [7]. To analyze cyber threats based on traffic profiles of SCADA hosts, it is therefore essential to take traffic characteristics of PLCs into account. For example, to detect botnets, frequency-domain analysis techniques, such as discrete Fourier transform (DFT), have been applied to discover the major frequencies resulted from the communication

between bot-infected hosts and command-and-control (C&C) servers (e.g., [8], [9], [10]). In [10], flow-specific packet sizes and packet inter-arrival timestamps within a configurable time window are collected to create time series for DFT analysis. To leverage frequency-domain features and develop algorithms detecting cyber threats such as botnets in SCADA environments using co-simulators, it is indispensable to incorporate accurate traffic models of PLCs into co-simulation platforms.

In this work, we model the traffic characteristics of PLCs and establish a parameterized model that can be incorporated into existing co-simulation platforms. We focus on the distributed network protocol (DNP3) [11], which is widely used in existing SCADA systems in CPES. Taking the execution model of PLCs into account, we propose to extract necessary model parameters from PLC program and configuration files to determine packet sizes. In addition, we take measurements from real PLCs to find proper model parameters for timing characteristics. Currently, we focus on PLCs working as DNP3 slaves, which are widely used in SCADA systems of CPES. As DNP3 masters run sophisticated applications and can be implemented on various devices such as high-end PLCs and computers (e.g., using the Opendnp3 library [12]), the traffic model for DNP3 masters is not discussed in this work. To validate our approach, we construct a traffic model for a real PLC running the DNP3 protocol as a slave node and incorporate it into OMNeT++. Simulated traffic data is then compared against network traces generated by another PLC running the same program. Our evaluation shows that the proposed approach captures essential traffic characteristics of PLCs, which can be used to extend existing co-simulation platforms and gain further insights into the behaviors of the smart grid as an entirety.

II. State of the Art and Limitations

The smart grid is a complex system consisting of interdependent cyber and physical components. Co-simulation is an effective way of analyzing the behaviors of a CPES because interactions among its multiple cyber and physical components are jointly modeled and simulated. To perform integrated analysis, simulators from different domains can be leveraged in co-simulation. For example, the INSPIRE co-simulation environment [2] is a hybrid framework leveraging the high-level architecture (HLA) and combining a continuous-time power-system simulator (i.e., DIgSILENT PowerFactory), a discrete-event network simulator (i.e., OPNET), and power-system control and protection algorithms implemented in general programming languages such as MATLAB and Java. By implementing logical time synchronization strategies among its domain-specific components, INSPIRE allows us to study the impacts of communication delays on real-time smart-grid monitoring. Emulation tools can also be exploited by co-simulation platforms. To study the impact of cyber attacks such as TCP SYN flood attack, a co-simulation testbed with hardware-in-the-loop capability is developed in [13]. In [14], the CORE emulator is used to model a SCADA network on a single computer. However, the traffic traces generated by existing co-simulation platforms are not realistic for applications such as traffic-monitoring-based intrusion detection since the distinction between field devices and computing servers is overlooked.

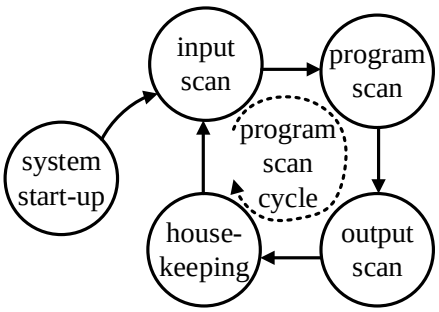


Fig. 1. Program scan cycle of a PLC.

Although communication protocols specific to SCADA network of CPES are implemented in different co-simulation platforms, PLC system configuration parameters related to traffic characteristics are yet to be incorporated. For instance, INSPIRE implements IEC 61850 and object linking and embedding (OLE) for process control (OPC), and communication delays introduced by packet transmission over SCADA networks are modeled by propagation delays of communication links. However, packet processing time induced by PLCs are not taken into account. To study frequency-domain features of PLCs, especially in the context of cyber-threat detection (e.g., [10]), it is important to properly incorporate the impacts of the execution model of PLCs on traffic timing characteristics, e.g., by introducing additional timing parameters into the network simulator/emulator.

We note that the execution model of PLCs has been leveraged in the design of the co-simulation framework proposed in [3] to find the proper time step size of co-simulation, balancing between computation complexity and simulation accuracy. However, only the length of the program scan cycle is taken into account in [3]. To generate accurate and realistic network traces of SCADA networks, a more detailed model of PLCs is yet to be developed for CPES co-simulation tools.

III. Background – Ladder Logic Execution Model

To control a power system physical process, a PLC (e.g., intelligent electronic device considered in the microgrid system in [3]) is designed to execute its control program as a control loop. Each iteration of the control loop is a program scan cycle. The time it takes to complete a program scan cycle is known as the scan cycle time. As illustrated in Fig. 1, each program scan cycle consists of four phases [15] after system start-up. In the input scan phase, status of all the input terminals used by the control program is copied into the input image, which is typically organized as a table of input words/bits. In the program scan phase, the PLC control program is executed using the data collected during the input scan phase. The resulting logic is written into the output image, which is then transferred to output terminals in the output scan phase. Finally, internal checks on memory, execution speed,

and operation status are performed during the housekeeping phase. Communication requests are also processed during this phase: Packets received from the SCADA network are properly parsed and serviced, and those generated by the PLC during previous phases are transmitted. After the housekeeping phase is completed, the PLC firmware moves on to perform input scan for the next program scan cycle.

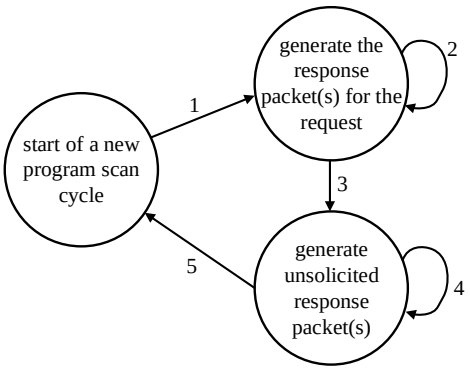
The actual scan cycle time depends on a variety of factors [15], including processor speed, control program size, the types of instructions executed, and the actual true/false conditions monitored by the control program. At the end of each program scan, PLC firmware calculates the actual scan cycle time and stores it in the system file. It is possible to monitor scan time information (e.g., maximum scan cycle time and the last scan cycle time) via PLC programming. In contrast to computers running desktop operating systems, the execution model of PLCs leads to a different traffic characteristics: Communication requests are serviced during the last phase of each program scan cycle, and the communication workload will impact the actual scan cycle time. In addition, it should also be noted that a PLC is not able to properly react to input signals that change at rates close to (or even higher than) its program scanning rate. In our proposed PLC traffic model, we thus assume that PLCs deployed in a CPES are fast enough to always keep up with the rates of input changes.

IV. Parameterized Traffic Model for DNP3 Slave PLCs

As discussed in Sec. I, modeling the traffic characteristics of PLCs enables us to exploit network traces generated by co-simulation in applications relying on realistic traffic profiles, such as the design of a botnet detection algorithm based on network monitoring [10]. In this section, we describe our proposed DNP3 traffic model for PLC slave nodes, which can be incorporated into the network simulator/emulator of a co-simulation platform by creating a PLC class. In network simulators such as OMNeT++ [16], this can be achieved by sub-classing the class representing a generic network host. To generate realistic SCADA traffic profiles, two sets of control parameters (i.e., class attributes) need to be added to the PLC class to model its communication architecture, namely packet-size parameters and time parameters. By simulating the communication architecture of PLCs, the packet processing routine (e.g., `handleMessage()` in OMNeT++) controls the network behaviors of simulated PLCs according to its control parameters.

A. Communication Architecture of PLCs

The communication architecture of a PLC is comprised of three major components, i.e., communication instruction/configuration, communication queues, and communication request queue. If a PLC initiates the communication with another SCADA host, it sets up read/write request messages using PLC communication instruction (e.g., the `MSG` command that prepares a message for transmission). The message generated by a communication instruction is placed into the transmit queue. If many packets are generated during the program scan phase, those that cannot be accommodated by the transmit queue are buffered in the communication request queue, which works in first-come first-served (FCFS) mode. Whenever a buffer in the transmit queue becomes available, the first message request in the communication request queue will be processed and the resultant message will be placed in the transmit queue. Note that unlimited request buffering is supported for the transmit queue of many PLCs, so the number of packets that can be generated during a program scan is not restricted. When a PLC processes requests from another host (e.g., another PLC or computer serving as the master), it places requests from the master into its receive queue. Note that the receive queue has a limited size and request buffering is not available. Request message from the master is automatically parsed and processed by the PLC's firmware: If the read/write operation specified by the master is supported by the PLC, it will process the packet by reading/writing proper PLC files and/or generating a proper response message. In the rare scenario where the number of requests from the master exceeds the size of the receive queue, those that cannot be buffered into the receive queue will not be served during the current program scan cycle. If the master is not configured to re-send the request, a communication failure occurs, which can be monitored by reading its communication status bits.



state transition conditions

1. Request(s) are available to be serviced or unsolicited response(s) need to be generated
2. There are still request(s) to be serviced
3. There is no request to be serviced
4. There are still unsolicited response(s) to be generated
5. There is no unsolicited response to be generated

Fig. 2. State machine for DNP3 response packet generation at PLCs.

To model such a communication architecture in network simulation for PLCs working as DNP3 slaves, only the size of the receive queue needs to be added to the PLC class as an attribute. We denote the receive queue size by N_r. Such information can be found in documentation on PLC instruction set and hardware architecture. For instance, according to [5], the MicroLogix 1400 PLC has 8 buffers in its receive queue, i.e., $N_r = 8$. The state machine shown in Fig. 2 can be incorporated into the packet processing routine of the PLC class: If a DNP3 slave receives request(s) from its DNP3 master, it parses the request and performs the

specified operation. In co-simulation environments, contents of the packets may be generated by other domain-specific tools. For example, in [2], voltage and current sensor readings are collected from power system simulators, while system commands are generated by control applications written in general programming languages. For read request, data can be queried from power-system simulator to prepare the response message. For write request, data can be sent to power-system control application and the execution status (e.g., return values from a control application) needs to be collected to form the response message. In the case where unsolicited response is enabled, power-system simulator needs to be queried to detect system events (e.g., status changes of input terminals). Note that this state machine needs to be executed at a time step that matches the scan cycle time.

B. Packet-Size Parameters

Open-source implementation of DNP3 protocol, such as the Opendnp3 library [12], can be ported into network simulators to generate packets that conform to the DNP3 packet layout (e.g., [17]). However, the following parameters still need to be extracted from a PLC's DNP3 configuration to simulate its traffic characteristics:

1) *Maximum response size*. A PLC sends DNP3 application layer frame to fit into the pre-specified maximum response size, which is denoted by $S_{\max}$. If the size of a response message exceeds $S_{\max}$, it will be fragmented into multiple response packets for transmission.
2) *DNP3 data object and configuration parameters.* When configured as a DNP3 slave node, the mappings between data elements in PLC data files (e.g., input file and counter file) and DNP3 data objects of class levels $0 \sim 3$ need to be specified. For instance, to link a PLC binary input file with the DNP3 binary input object, the identifier of the binary input file (e.g., an integer file number) is used. In addition, a binary input object configuration file is used to specify the class levels of individual elements in the binary input file. If unsolicited response is enabled for a particular class level, the corresponding DNP3 protocol configuration bit will be set and a threshold on the number of events will be specified.

These parameters allow us to determine the response packet sizes when a request from a DNP3 master is received. For instance, suppose that a poll request for class 1 and class 2 events is received by the PLC. It then retrieves events for DNP3 data objects of these two class levels. The events (e.g., new input terminal status) are then used to generate a DNP3 response. In the case where unsolicited response for a particular class level is enabled, the PLC generates an unsolicited response when the number of queued events reaches the specified threshold. If the length of the response is larger than $S_{\max}$, it is fragmented into multiple response packets. These parameters can be extracted from PLC program and configuration files and then added as attributes of the PLC class. In fact, when the control scheme based on DNP3 is developed (i.e., before its implementation in PLCs), such parameters are already available. In co-simulation tools modeling network behaviors and control logic of SCADA hosts separately (e.g., [2]), these parameters may be shared between the network simulator and the control applications.

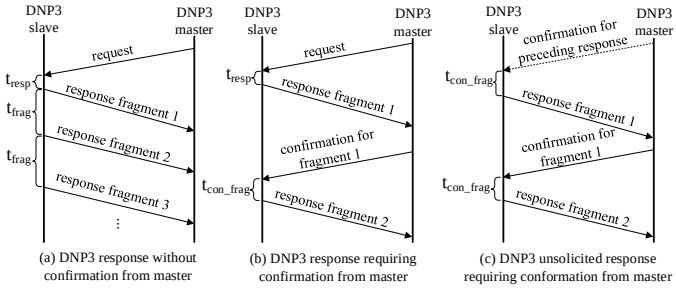


Fig. 3. Parameters modeling the response preparation time of DNP3 slave PLCs.

C. Time Parameters

To model the interactions between a DNP3 slave PLC with its master node, we need to consider the following parameters:

1) *Response preparation time*. As a DNP3 slave may generate responses upon requests and/or unsolicited responses, three timing parameters shown in Fig. 3 need to be introduced. For response packets generated upon a master request that do not require confirmation from the master (e.g., a poll request on class 0 data objects), t_{resp} is the time it takes for the slave PLC to process the request and send out the first response fragment. In addition, parameter t_{frag} is introduced to model the time spacing between two consecutive DNP3 response fragments. For solicited/unsolicited response packets requiring confirmation from the master, $t_{\text{con\_frag}}$ is the time it takes for the slave to process the confirmation from master and send out the next response fragment. Note that it is possible that the response fragment preceding the first unsolicited response fragment shown in Fig. 3c does not require confirmation. In this case, the time between these two fragments should be modeled by t_{frag}.
2) *Confirmation timeout and retransmission count*. When a response fragment requires confirmation from the master, the slave will not send the next response fragment until the confirmation is received. The confirmation timeout is used to determine when the slave should retransmit the fragment or abort operation. In co-simulation, if the network-induced delay for a certain fragment (e.g., propagation and queuing delays) is larger than the confirmation timeout, it should be deleted from the scheduled event list when the timeout event occurs. In addition, a retransmission may be performed if the retransmission count of the PLC is set to a non-zero value.
3) *Hold time after events*. This parameter is only for unsolicited responses, and it is combined with the threshold for class-specific events. If the threshold is reached or the time elapsed after the a class-specific event enters

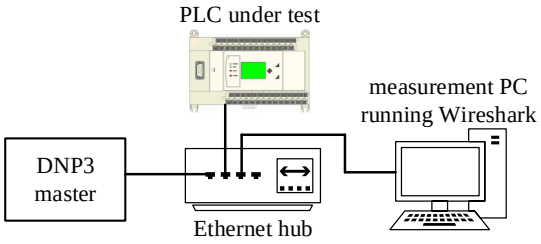


Fig. 4. Test bed for collecting network traces from a slave PLC.

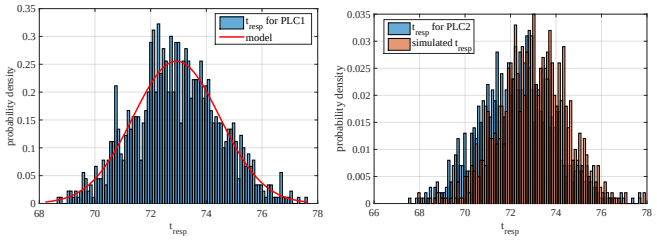
the queue reaches the specified hold time, unsolicited response for the particular class will be generated.

The time parameters and the interaction patterns illustrated in Fig. 3 are used to specify how a particular response generated according to the state machine in Fig. 2 should be processed. However, it should be noted that the execution of the state machine in Fig. 2 occurs in the housekeeping phase of the program scan cycle. In co-simulation platforms, solicited/unsolicited responses should be generated immediately when data/request from the power-system simulator or control application allows. The scan cycle time and the extra time incurred by packet processing during the housekeeping phase are modeled by t_{resp}. For instance, if the class 1 events collected from the power-system simulator trigger the generation of an unsolicited response, t_{resp} models the time between the moment the condition for generating this response becomes true and the moment its first fragment is sent.

D. Finding Time Parameters Through Measurements

As mentioned in Sec. III, the scan cycle time is not a constant. Although execution time of PLC instructions under different logic conditions (e.g., the actual true/false condition on a ladder logic rung) can be found in the documentation (e.g., Appendix A of [5]), it is hard to determine the time consumed by a particular program scan cycle without actually evaluating the logic conditions for all the PLC program instructions. Since our goal is to establish a traffic model capturing the time and packet size characteristics of DNP3 slave PLCs, we find the models for the time parameters t_{resp}, t_{frag} and $t_{\text{con\_frag}}$ by analyzing traffic traces taken from real PLCs. Fig. 4 shows the configuration of our test bed for collecting PLC network traces. An Ethernet hub (or an Ethernet switch with port mirroring properly configured) interconnects the DNP3 master, the PLC under test, and the measurement PC. Note that the DNP3 master in a real SCADA system may be implemented by high-end PLCs or even a computing server because it needs to process a large number of solicited/unsolicited responses.

To measure t_{resp}, we configure the master to generate DNP3 poll request for class 0 data objects and find the time it takes for the slave to send out the first fragment of its response. Note that the PLC under test needs to be configured to disable confirmation for its responses. Similarly, t_{frag} can be measured by analyzing the network traces and finding the time spacing between consecutive response fragments requiring no master confirmation. To measure $t_{\text{con\_frag}}$, we enable class 1 and class 2 events in the PLC under test and adjust the maximum response size so that unsolicited responses are fragmented into multiple packets. Using network traces captured by the measurement PC, we find $t_{\text{con\_frag}}$ by analyzing the time spacing between fragments for the same unsolicited response and their confirmation messages from the master.



(a) Normalized probability distribution of the time parameter t_{resp} of PLC1 and its fitted normal distribution model

(b) Normalized probability distribution of the time parameter t_{resp} of simulated PLC vs. that of PLC2

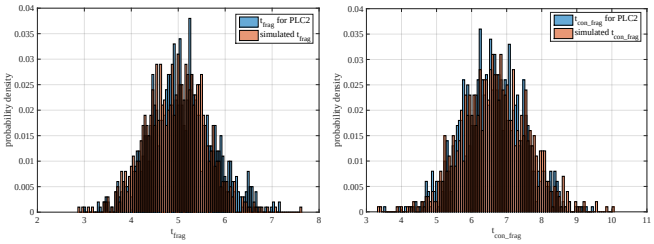
Fig. 5. Model for t_{resp} constructed based on network traces from PLC1 and simulated PLC traces compared against those from PLC2.

V. Model Construction and Evaluation

To evaluate the quality of our proposed DNP3 traffic model for slave PLCs, we use our test bed to collect traffic data from a MicroLogix 1400 PLC (PLC1). A time parameter is modeled as a random variable and its samples are extracted from the network traces collected by the measurement PC running Wireshark. A proper distribution is fitted to the collected samples, which is then incorporated into OMNeT++ to generate simulated network traces. Then, we replace the PLC under test with another PLC of the same model (PLC2) running the same PLC program with exactly the same configuration. The network traces collected from this PLC are then compared against the simulated traces to see whether they have similar distributions. We configure PLC under test as a datagram end point so requests from the master and solicited/unsolicited responses from the slave are all transmitted via UDP.

A. Modeling t_{resp}

Before data collection is initiated on the measurement PC, we first adjust the rate at which DNP3 requests are generated to match the program scan cycle time such that a single request is processed during each program scan cycle. The PLC program of the slave consists of a set of 100 counters. The input logic condition of all the counters are tied to the same input terminal, which is connected to a function generator. During each program scan cycle, a false-to-true transition is fed to the input terminal, triggering the counters to increment by 1. When a request is received, the PLC under test collects all counter values and generate a DNP3 response. We find that a normal distribution can be fitted to the samples of t_{resp}. For the particular PLC program, we have $t_{\text{resp}} \sim \mathcal{N}(72.9\mu\text{s}, 2.4\mu\text{s})$. Fig. 5a compares the fitted distribution with the actual data collected from PLC1. The distribution of t_{resp} of simulated PLC traces is compared against that of the data collected from PLC2 in Fig. 5b. The two distributions resemble each other, indicating that the proposed measurement-based approach is suitable for modeling t_{resp}.



(a) Normalized probability distribution of the time parameter t_{frag} of simulated PLC vs. that of PLC2

(b) Normalized probability distribution of the time parameter $t_{\text{con\_frag}}$ of simulated PLC vs. that of PLC2

Fig. 6. Models for t_{frag} and $t_{\text{con\_frag}}$.

B. Modeling t_{frag} and t_{con_frag}

To measure t_{frag} and $t_{\text{con\_frag}}$, we set the maximum response size to 50 bytes to generate multiple response fragments. Note that class 1 and class 2 events are enabled for the counter objects to generate unsolicited responses, with each class level assigned to 50 counter objects. Fig. 6a compares the distribution of t_{frag} of the simulated PLC against that of PLC2. The normal distribution $\mathcal{N}(4.9\mu\text{s}, 0.4\mu\text{s})$ is used as the model for t_{frag}. Fig. 6b compares the distribution of $t_{\text{con\_frag}}$ of the simulated PLC against that of PLC2. The normal distribution model for $t_{\text{con\_frag}}$ is $\mathcal{N}(6.7\mu\text{s}, 0.8\mu\text{s})$. We observe that the constructed models for t_{frag} and $t_{\text{con\_frag}}$ are good approximations to the actual distributions. Combining the time parameters models, our traffic model generates simulated packet traces with timing characteristics closely resembling those of real PLC traffic.

C. Computing Packet Sizes

In our simulation, we incorporate packet-size parameters (see Sec. IV-B) into the PLC class without implementing the DNP3 protocol. As a result, only the packet sizes generated by the simulation are compared against the PLCs traces collected from our test bed. We find that our proposed model can be used to generate accurate packet size information. We note that, however, this is possible because the PLC program used in our evaluation is simple enough to analyze. For the co-simulation of a realistic CPES, additional information that impacts the packet sizes, such as the number of events generated during each program scan cycle, can be obtained from the power-system simulator.

VI. Conclusion

In this work, we propose a traffic model for PLCs serving as DNP3 slaves that can be incorporated into existing co-simulation platforms to generate more realistic network traces suitable for evaluating applications relying on traffic characteristics of SCADA hosts, such as cyber threat detection algorithms based on traffic monitoring (e.g., [10], [8], [9]). To correctly compute packet sizes, we identify parameters that need to be extracted from DNP3 slave configuration and incorporated into network simulation. To model the timing characteristics of DNP3 slave PLCs, we introduce time parameters and propose that proper models for them can be established via measurements. We evaluate our proposed approach using real PLCs. Our evaluation results show that the proposed approach establishes realistic traffic model for PLCs.

As our future work, we will enhance our proposed approach and model other end point types supported by PLCs (e.g., listening end point and dual end point [5]). In addition, we will also consider extending our model for DNP3 masters running on different types of devices, such as computing servers and high-end PLCs.

Acknowledgment

The material is based upon work supported by the Department of Energy under Award Number DE-OE0000779.

References

[1] "Programmable Controllers - Part 3: Programming Languages," IEC 61131-3:2013, International Electrotechnical Commission (IEC), February 2013.

[2] H. Georg, S. C. Mller, C. Rehtanz, and C. Wietfeld, "Analyzing Cyber-Physical Energy Systems: The INSPIRE Cosimulation of Power and ICT Systems Using HLA," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 4, pp. 2364–2373, November 2014.

[3] V. Kounev, D. Tipper, M. Levesque, B. M. Grainger, T. Mcdermott, and G. F. Reed, "A Microgrid Co-simulation Framework," in *Proceedings of the 2015 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*, April 2015, pp. 1–6.

[4] D. M. E. Ingram, P. Schaub, R. R. Taylor, and D. A. Campbell, "Performance Analysis of IEC 61850 Sampled Value Process Bus Networks," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 3, pp. 1445–1454, August 2013.

[5] "MicroLogix 1400 Programmable Controllers Instruction Set Reference Manual," *Rockwell Automation Publication 1766-RM001F-EN-P*, May 2014.

[6] D. Zhao, I. Traore, B. Sayed, W. Lu, S. Saad, A. Ghorbani, and D. Garant, "Botnet Detection Based on Traffic Behavior Analysis and Flow Intervals," *Computers & Security*, vol. 39, Part A, pp. 2–16, 2013.

[7] R. R. R. Barbosa, R. Sadre, and A. Pras, "Difficulties in Modeling SCADA Traffic: A Comparative Analysis," in *Proceedings of the 13th International Conference on Passive and Active Measurement (PAM)*, 2012, pp. 126–135.

[8] J. Kwon, J. Lee, H. Lee, and A. Perrig, "PsyBoG: A Scalable Botnet Detection Method for Large-Scale DNS Traffic," *Computer Networks*, vol. 97, pp. 48–73, March 2016.

[9] G. Bottazzi, G. F. Italiano, and G. G. Rutigliano, "Frequency Domain Analysis of Large-Scale Proxy Logs for Botnet Traffic Detection," in *Proceedings of the 9th International Conference on Security of Information and Networks (SIN '16)*, July 2016, pp. 76–80.

[10] P. Narang, C. Hota, and H. T. Sencar, "Noise-Resistant Mechanisms for the Detection of Stealthy Peer-to-Peer Botnets," *Computer Communications*, vol. 96, pp. 29–42, December 2016.

[11] "IEEE Standard for Electric Power Systems Communications - Distributed Network Protocol (DNP3)," *IEEE Std 1815-2012 (Revision of IEEE Std 1815-2010)*, pp. 1–821, October 2012.

[12] *Opendnp3 project, https://github.com/automatak/dnp3*.

[13] R. Liu and A. Srivastava, "Integrated Simulation to Analyze the Impact of Cyber-Attacks on the Power Grid," in *Proceedings of the 2015 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*, April 2015, pp. 1–6.

[14] V. Venkataramanan, A. Srivastava, and A. Hahn, "Real-Time Co-Simulation Testbed for Microgrid Cyber-Physical Analysis," in *Proceedings of the 2016 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*, April 2016, pp. 1–6.

[15] F. Petruzalla, *Programmable Logic Controllers, 5th Ed.* McGraw-Hill Education, January 2016.

[16] A. Varga, "OMNeT++ Simulation Manual Version 5.0," *Available at: https://omnetpp.org/doc/omnetpp/manual/*, 2016.

[17] C. Queiroz, A. Mahmood, and Z. Tari, "SCADASim - A Framework for Building SCADA Simulations," *IEEE Transactions on Smart Grid*, vol. 2, no. 4, pp. 589–597, December 2011.