Integrated Software Testbed for Cyber-Physical Analysis in Smart Grid

Song Tan, Wen-Zhan Song
Department of Computer Science
Georgia State University
Atlanta, GA, USA
stan6@student.gsu.edu, wsong@gsu.edu

Lang Tong
School of Electrical and Computer Engineering
Cornell University
Ithaca, NY, USA
ltong@ece.cornell.edu

Yuan Wu
College of Information Engineering
Zhejiang University of Technology
Hangzhou, China
iewuy@zjut.edu.cn

Abstract—Smart Grid is a highly complex cyber-physical power system that involves a huge amount of embedded devices for sensing, control, computation and communication. To validate the functionality, security and reliability of such a system requires the modeling and emulation of both power network and communication network, as well as the interactions between them. In this paper, we present the design, implementation and evaluation of an integrated software testbed for cyber-physical analysis in Smart Grid. Compared with previous related work, our software testbed fills the gap by: 1) employing emulation features such that the critical control programs tested in our testbed can be directly ported to physical embedded devices; 2) supporting both real time and non-real time analysis by implementing virtual time emulation; 3) public distribution as an open source software. A walk-through study case within the advanced metering infrastructure is presented to demonstrate the comprehensive support for smart grid applications.

Index Terms— cyber physical; real time; smart grid; software testbed

I. INTRODUCTION

Smart Grid is a highly complex cyber-physical power system characterized by the two-way communications of data and control signal, large-scale penetrations of renewable energy, and the complex interactions among distribution systems, energy markets, and customer behaviours. Due to the system complexity, typically the high cohesion of communication and power system, validating new ideas in Smart Grid applications such as distributed control algorithms, cyber-security strategies and marketing policies are difficult, especially in a lab environment.

The existing works to solve the problem are generally focused on two categories: hardware platforms and software simulation platforms. The hardware platform approach achieves high fidelity by employing dedicated devices as part of the testbeds. Stanovich et al. [1] and Hahn et al. [2] did a good job by integrating dedicated devices, such as Remote Terminal Unit and Ethernet switch within the testbeds, to achieve the hardware-in-the-loop testing. However, the problems with the hardware platforms are that since the dedicated hardware are the integral parts of the testbeds, they cannot be easily accessed and used by the public research community and difficult to be scaled when the test case becomes quite large. Software simulation platforms, on the other hand, achieve better availability, usability and scalability. They usually combine multiple simulation tools, typically a network simulator and an electric power grid simulator, and a middleware is used to exchange messages periodically and synchronize all the simulators. In [3], Hopkinson et al. introduce a federated simulation combining NS2, a discrete event network simulator with PSCAD, a continuous time power network simulator. In [4], Godfrey et al. simulate the Smart Grid using NS2 and OpenDSS, a power network simulator. Within simulation platforms, the models of various objects can be easily scaled and statistically analyzed. However, since simulation typically abstracts the operating system, communication protocols and power dynamics into various mathematical simulation models, it can only duplicate the behaviour and structure of the system, but not the execution environment of critical control programs.

Different from the above related work, we adopt the software emulation approach in our platform. Software emulation achieves high fidelity by duplicating the code execution environment of critical control program in each virtual node, such that the programs tested in the emulation platform can be directly ported to the embedded devices as firmware [5]. As the first emulator for Smart Grid, our previous work in [5] successfully achieves the features by using Linux namespace, a recent light weighted paravirtualization technique supported by mainstream Linux kernel. Each emulated virtual node is a complete and independent Linux environment such that the program running in it behaves exactly the same as it is ported to Linux enabled embedded devices. However, the problem is that the system clock can only be advanced at the same pace with the time of host operating system, in other words, the real time. Although our real time feature enables it to interact with physical devices, as the way works in [1] [2] with Real Time Digital Simulator, when a test case spans long period of time, such as several months or years, it suffers from being unlikely to complete the test case within a reasonable period of time. Moreover, in order to achieve time synchronization, its integration with other simulation platforms requires the simulators to be real-time capable, otherwise a time drift will occur to corrupt the result. Therefore, implementing controllable virtual time (non-real time) feature to the emulation platform is the way to tackle this issue. Zheng

This research is supported by funding NSF-CPS-1135814.
et al. implemented controllable virtual time to emulation and integrate the emulation platform with a parallel simulator in [6], but they use OpenVZ paravirtualization technique and only focus on sim/emulation integration of communication network.

In this paper, we present an integrated software emulation testbeds for Smart Grid. Specifically, by introducing virtual time to CORE [7], a Linux namespace based real time network emulator and integrating it with GridLAB-D [8], a popular open source power simulator that can work in both real time and non-real time mode, we fill the gaps of smart grid testbed research by:

- Employing emulation features such that the critical control programs tested in our testbed can be directly ported to physical embedded devices controlled by Linux operating system.
- Supporting both real time and non-real time analysis by introducing virtual time emulation such that users can choose the working mode according to the experiment scenario. User can choose real time mode when they want to conduct hardware-in-the-loop testing but choose non-real time mode when they simply want to complete the pure software test case as fast as possible.
- Public release as open source software in https://sourceforge.net/projects/scoreplus/ for user access and further development.

II. SYSTEM DESIGN AND IMPLEMENTATION

Figure 1 illustrates the whole architecture of our platform and the integration approach of the GridLAB-D power simulator with virtual-time based CORE. The power simulator is responsible for constructing the grid topology and the power physics models of various objects in electrical power network, such as transformer, solar, power meters, etc. Each of these object in power simulator is corresponded with one Virtual Node (VN) in the communication network emulator, in which the control program for that object is running. All the VNs are connected within wired/wireless communication network so that they can communicate with each other by sending and receiving packets.

A. Global synchronization

Our platform can work in both real time mode and non-real time mode. In real time mode, since CORE is originally a real time emulator, it can be synchronized directly with the real time mode GridLAB-D. In non-real time mode, the synchronization module bridges the power simulator and the communication emulator together by managing two types of messages: interactive messages and synchronization messages. Interactive messages are initiated by VNs through sockets to set/get the property values of the corresponding simulated objects in power simulator. All the interaction messages and the responses between one VN and the corresponding simulated object are managed independently by one thread within the message broker, so that they can be cached and injected at the specific times which will be defined later. Synchronization messages are employed to achieve the conservative parallel/distributed simulation synchronization [9] between the power simulator and the communication emulator. The global clock in synchronization module controls the advancing of time through Sync_win and both the simulation clock and the virtual clocks of emulation will be aligned to the global clock after each iteration of synchronization. The specific execution logics of synchronization module are abstracted in Algorithm 1.

B. Virtual time of emulation

The virtual time of emulation is implemented within the Virtual time control component in Figure 1. The significance of introducing virtual time to the communication emulator is two-fold. Firstly, the clock of the power simulator is discrete and advances in various speed driven by events, in order to synchronize the emulation with the simulation platform, we have to replace the original fixed real time clock with a controllable virtual time clock, such that the synchronization can be achieved using Algorithm 1. Secondly, we have observed from our experiments that the VNs of the emulator are sometimes in idle situations, which means the programs running inside the VNs don’t or cannot change the state of the system within a particular period of time. These situations include no running process within a VN, process is executing sleep() function, and process is trying to interact with the power simulator through socket (which has to be blocked and only returned at the beginning of the next Sync_win). By introducing virtual time to emulation, we can jump the virtual clocks directly when the VNs are idle, which is the key point to advance the clock as fast as possible. Algorithm 2 illustrates the details of the virtual time control. It is a straight variation of barrier synchronization algorithm [6] [9] since the virtual clocks of all the VNs should be synchronized.

III. CASE STUDY

The case we created is based on the AMI network test case from American Electric Power Company [10] and the IEEE...
Algorithm 1 Global Synchronization
1: \( t_{sim} = t_{emu} = t_{global} = 0 \)
2: if in real-time mode then
3: repeat
4:  Concurrently run GridLAB-D power simulator and CORE in real time mode.
5: until \( t_{global} \neq END\_TIMESTAMP \)
6: else
7: while \( t_{global} \neq END\_TIMESTAMP \) do
8:  \( t_{emu} = t_{emu} + \text{Sync}_\text{win} \)
9:  if \( t_{emu} \neq t_{global} \) then
10:      Inject the responses of interactive messages within last \( \text{Sync}_\text{win} \) to the corresponding virtual nodes.
11:      Run all virtual nodes for \( \text{Sync}_\text{win} \) (Algorithm 2)
12:  \( t_{emu} = t_{emu} + \text{Sync}_\text{win} \)
13:  Collect and cache all the interactive messages from all the virtual nodes within current \( \text{Sync}_\text{win} \).
14: end if
15: if \( t_{sim} \neq t_{global} \) then
16:      Inject all the cached interactive messages to the power simulator and cache the responses.
17:      Run the power simulator for \( \text{Sync}_\text{win} \).
18:  \( t_{sim} = t_{sim} + \text{Sync}_\text{win} \)
19: end if
20: end while
21: end if

Algorithm 2 Virtual time control of emulation
1: \( \text{barrier} = t_{emu} + \text{Sync}_\text{win} \)
2: for all \( VN_k \) do
3:  \( VN_k.isComplete = false \)
4:  \( VN_k.stop_T = \text{barrier} - VN_k.offset \)
5:  if \( VN_k.stop_T < VN_k.sleep_time \) then
6:      \( VN_k.sleep_time = VN_k.sleep_time(VN_k.stop_T - VN_k.clock) \)
7:      \( VN_k.clock = \text{barrier} \)
8:      \( VN_k.offset = 0 \)
9:  \( VN_k.is\_runnable = true \)
10: else
11:      \( VN_k.clock = VN_k.clock + VN_k.sleep_time \)
12:      \( VN_k.sleep_time = 0 \)
13:      \( VN_k.is\_runnable = true \)
14: end if
15: while \( VN_k.isComplete = false \) 
16:  Assign a timeslice of real time to \( VN_k \) to run
17: repeat
18:  Advance \( VN_k.clock \) (same speed as real time)
19: until The timeslice elapses
20: if \( VN_k \) is idle (eg. No running process, sleep(\( sleep\_time \)), or tries to interact with power simulator through socket) then
21:      \( VN_k.offset = 0 \)
22:      \( VN_k.clock = \min(VN_k.stop_T, VN_k.clock + VN_k.sleep_time) \)
23:      \( VN_k.sleep_time = \max(0, VN_k.clock + VN_k.sleep_time(VN_k.stop_T - VN_k.clock)) \)
24: end if
25: if \( VN_k.clock \geq VN_k.stop_T \) then
26:      \( VN_k.offset += VN_k.clock - \text{barrier} \)
27:      \( VN_k.clock = \text{barrier} \)
28:      \( VN_k.isComplete = true \)
29: end if
30: end while
31: end for

Fig. 2. The Study Case

PES 37 bus distribution system test feeders [11]. Figure 2 shows the building blocks of our experimental scenario:

- Operation layer: The control program in control center broadcasts real-time energy prices every 5 minutes and also collects meter reading data through AMI Head-End, which is the gateway to the AMI network. Meanwhile, the control center also calculates the bills for each house, based on the real time price and the collected energy consumption data.
- Customer layer: The IEEE 37 bus distribution test feeders is set up to provide power for 200 residential houses. Each house is equipped with loads including a water heater, a dryer, a PHEV, a solar panel, and a storage. Moreover, a smart meter is employed to serve as the interface between the power network and AMI for each house. The program running in smart meter responses to the real time prices. Using interactive messages, the smart meter program adjusts the setpoint of appliances within each house correspondingly based on the price-responsive control model in [12]. It also measures the energy consumption of each house and sends the data to the control center through AMI network on a hourly basis.
- AMI network: AMI enables communications and interactions between/within the operation layer and the customer.
layer. The control center and AMI Head-End is connected through Internet. AMI Head-End, the relay nodes and the smart meters are formed as a IEEE 802.11 Radio Frequency Mesh network.

We emulate and analyze the behavior of the whole system within one day in both real time mode and non-real time mode. In real time mode, the control center is served by a physical HP PC running Ubuntu 12.04, which interacts with our platform through Internet using a public IP address. In non-real time mode, the control center is simply a emulated virtual node. The same control program is running in both cases. As shown in Figure 3, when we conduct the experiment in real time mode, it takes about 24 hours to complete the test case. In non-real time mode, the same test case can be completed in about 10.4 hours. Figure 4 shows the energy consumption of a water heater versus the real time energy prices over a day. We can see that the control program for the price responsive model in [12] for energy consumption tends to shift the energy consumption to the lower price period of the day.

![Fig. 3. Clock proceeding speed](image)

![Fig. 4. The water heater energy consumption and real time energy price](image)

Suppose the customer under smart meter X wants to manipulate his energy bill without being caught. In order to achieve this, he launches a Distributed Denial-of-service attack to the bi-direction data flow within AMI, which consists of the energy consumption data from the smart meters to the AMI Head-End, and the energy price data from the AMI Head-End to the smart meters. For the energy consumption data, the attacker modifies the ones from smart meter X and his targeted neighbors, such that each targeted neighbor has an increase in the reported energy consumption compared to the actual consumption, and the smart meter X has a decrease equal to the total increase of its targeted neighbors in the reported energy consumption. In this way, from the perspective of utility company, the total energy provided still conforms to the total energy being billed. For the energy price data, the attacker modifies the price to a lower value, such that based on the price responsive model in Figure 4, the actual energy consumption of each targeted neighbors will also increase. In this case, from the perspective of each targeted neighbor, the minor increase in the reported energy consumption data due to attack will become even less noticeable.

Specifically, as shown in Figure 5, the customer of smart meter X attacks three relay nodes at the same time: its own direct cluster head (Relay 2) and two neighbor cluster heads (Relay 1 and 3). Originally, Relay 1 and Relay 3 will directly interact with Relay 4 for the bi-direction data. We can see this from the result of `route` command in the terminal of Relay 4.
Relay 1. To reach 192.169.0.4, which is the IP address of Relay 4, no intermediate gateway is needed and packets can be simply forwarded through interface eth0. However, after attack, there is one extra high priority entry in the routing table of Relay 1 such that the packets designated to 192.169.0.4 will be forwarded to 192.169.0.2 first instead of the original one hop reach. As a result, for Relay 2, besides the data packets of the 10 customers within its own cluster, it will also intercept the data packets of the other 20 customers within the clusters of Relay 1 and 3. By making the three Relay nodes working in concert to compromise the data, customer X could dramatically reduces its own reported energy usage. As shown in Figure 6, for smart meter X, even though the actual energy usage across the day is 64kwh, the reported data is manipulated to 35kwh. The remaining 64 − 35 = 29kwh are evenly added to the other 29 customers’ reported data. In this way, from the perspective of utility company, the total energy consumed still conforms with the total energy being billed. Moreover, from the perspective of each of the other 29 customers’, since only 29/29 = 1kwh is added to their energy consumption, which usually results in about 0.1$ increase in their bills, it is very much likely that the customer will just let it go. Also note that since the energy price is modified to a lower value after the attack, the real power consumption paradigm of the attacked neighbors changes dramatically, compared to the normal situation when the correct real time energy price is given. As shown in Figure 7, the real power consumption of the attacked neighbors stays at a relatively higher level all the time after the attack and the demand response through real time pricing is not working any more. If more neighbors are involved in the attack, this will severely increase load of the system, which can result in a higher cost of power transmission or even an outage. An effective approach to detect this kind of attack is by monitoring the network traffic. As shown in Figure 8, since the routing path of the packets is changed and much more data packets are forwarded to Relay 2, the throughput of Relay 2 will be increased unusually from the moment of attack. Also, the network traffic congestion at Relay 2 will result in an increase in the communication delay from Relay 1 to the AMI meter head.

![Fig. 6. Actual energy usage and reported energy usage after attack](image1)

![Fig. 7. The total real power consumption of the attacked neighbors](image2)

![Fig. 8. Throughput and Communication delay](image3)

IV. CONCLUSION

In this paper, we presented the design, implementation and evaluation of our integrated cyber-physical testbed for Smart Grid. We introduced virtual time to the emulation platform and accurately synchronize it with the GridLAB-D power simulator. From the study case, we fully demonstrated the capabilities of our platform to support various cyber physical experiments for Smart Grid applications. The software is successfully released as open source at https://sourceforge.net/projects/scoreplus/.

REFERENCES


