ScorePlus: A Software-Hardware Hybrid and Federated Experiment Environment for Smart Grid

Song Tan and Wen-Zhan Song, Georgia State University
Steve Yothment, Panasonic Automotive Systems of America
Junjie Yang, Shanghai University of Electric Power
Lang Tong, Cornell University

We present ScorePlus, a software-hardware hybrid and federated experiment environment for Smart Grid. ScorePlus incorporates both software emulator and hardware testbed, such that they all follow the same architecture, and the same Smart Grid application program can be tested on either of them without any modification; ScorePlus provides a federated environment such that multiple software emulators and hardware testbeds at different locations are able to connect and form a unified Smart Grid system; ScorePlus software is encapsulated as a resource plugin in OpenStack cloud computing platform, such that it supports massive deployments with large scale test cases in cloud infrastructure.

Categories and Subject Descriptors: D.2.5 [Testing and Debugging]: Testing tools

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: Smart Grid, Testbed

1. Introduction

With the recent advancement in monitoring, sensing, control and communication, plus the ever increasing penetration of renewable and distributed energy resources, the legacy power grid is now evolved along the journey to smart grid, which is a complex cyber-physical power system envisioned to achieve self healing, resilience, sustainability and efficiency. The Smart Grid innovation brings a lot of new research challenges to the agenda, such as secure communication protocol, anomaly detection and identification, real-time pricing scheme, and renewable resource integration, etc. Due to the system complexity, typically the high cohesion of communication and power network, validating and demonstrating these new ideas are not easy jobs, especially in a lab environment [Edgar et al. 2012].

In this paper, we present ScorePlus, a software-hardware hybrid and federated experimental environment whose design framework and major components can be extended to build other cyber-physical testing systems in general. ScorePlus fills the gap by the following key features:

This work is submitting to the Special issue on Emerging Technologies in Embedded Software and Systems.

Our research is partially supported by NSF-1125165, NSF-1135814, NSF-1303359, NSF-1442630, NSF-1066391 and NSFC-61202369.

Author’s address: Song Tan and Wen-Zhan Song, Department of Computer Science, Georgia State University; Email: stan6@student.gsu.edu, wsong@gsu.edu. Steve Yothment, Senior Electrical Engineer, Panasonic Automotive Systems of America, USA; Email: yothments@us.panasonic.com. Junjie Yang, School of Electric and Information Engineering, Shanghai University of Electric Power, China; Email: yangjj@shiep.edu.cn. Lang Tong, School of Electrical and Computer Engineering, Cornell University, NY, USA; Email: ltong@ece.cornell.edu.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2015 ACM. 1539-9087/2015/XXXX-ARTXXXX $15.00
DOI: http://dx.doi.org/10.1145/0000000.0000000

— **Software-Hardware Hybrid:** ScorePlus includes both software emulator and hardware testbed. The software emulator emulates the communication and energy behavior of Smart Grid entities. The hardware testbed hosts physical devices, such as micro renewable energy generators (solar panel, wind turbine), energy demanders, storages, and topology switches, which are all connected by power network and communication network in a physical environment. Both the software emulator and hardware testbed follow the same architecture and interface, and the same Smart Grid application program can be tested on either of them without any modification. With this integration, researchers may examine the performance of Smart Grid applications under realistic communication and computation constraints in hardware testbed, while evaluating the corresponding scalability in software emulator at the same time.

— **Federated:** ScorePlus enables federated and distributed experiments such that multiple software emulators and hardware testbeds running at different (local or remote) locations are able to connect and form a larger Smart Grid system. This makes it possible to scale up experiments without being constrained by the computation resources of a single software emulator server or the number of physical devices of a single hardware testbed. Upon specifying the connection interfaces between each other, each instance can capture its own interior system dynamics even without a prior knowledge of the complete Smart Grid topology. Meanwhile, each individual node, either virtual or real, can interact with any other nodes within the system to conduct a hardware-in-the-loop testing.

— **Integrated within cloud infrastructure:** ScorePlus software is encapsulated as a resource plugin within OpenStack cloud computing platform, such that it can be reused the same as any other resources in OpenStack cloud applications. In this way, ScorePlus software can be instantiated and launched on different computing, networking, and storage resources with different configurations to support massive deployments with large scale test cases, which are all based on the specifications in a OpenStack Heat template.

2. Related work

Smart Grid is an intelligent power system that involves various embedded devices for sensing, control, computation and communication. Validating the functionality, security and reliability of Smart Grid applications within such a system requires the modeling and emulation of both power networks and communication networks, as well as the interactions between them. The design and implementation of experiment environment for Smart Grid are challenging and have been studied for years. In this section, we conduct extensive survey about the previous related efforts, which can be summarized into two categories: real hardware testbed and software simulation.

2.1. Real hardware testbed approach

Real hardware testbeds are the platforms employing actual physical smart grid devices for the experiments. We further classify this line of works into two subcategories: flat-out hardware platforms and hardware-in-the-loop platforms.

2.1.1. Flat-out hardware platform: The flat-out hardware platforms are the ones which consist of pure hardware devices. As a grid scale, the Korean government selected the whole Jeju Island to build the Smart Grid testbed to allow the testing of Smart Grid technologies and business models [smart grid institute 2010]. In [Laboratory 2009], the Idaho National Lab incorporates the actual Smart Grid components including power generators, storage batteries, and substations to facilitate the cyber security research of power transmission in Smart Grid. In [Stimoniaris et al. 2011], Renewable Energy
Laboratory in Greece set up a central-controlled testbed consisting of PV-panels, battery banks and inverters to investigate the renewable integration issues. As a lab scale, the authors in [Song et al. 2012] design SmartGridLab testbed, which consists of intelligent power switch, power generator, renewable energy sources, smart appliances, and power meter, in order to test distributed demand response algorithm in Smart Grid. In [Lobo and Idowu 2015], Joyer et al. demonstrate a lab scale microgrid testbed, which is based on IEEE 1547 to serve as an interconnection standard.

2.1.2. Hardware-in-the-loop platform: In hardware-in-the-loop platforms, the hardware devices only serve as parts of platforms, and need to interact with other software simulations to conduct complete experiments. Hahn et al. in [Hahn et al. 2013] employs devices like Programmable Logic Units (PLUs) and Intelligent Electronic Devices (IEDs) for communication networks and Real-Time Digital Simulators for power network simulation. Stanovich et al. in [Stanovich et al. 2013] integrates hardware from energy field, such as Remote Terminal Unit (RTU), fiber optical cables within the testbed. Recently, the author in [Barzegaran 2015] employs devices like smart meters, phasor measurement units, phasor data concentrator, and hybrid vehicle charging system, as the essential components of microgrid testbed in lab.

2.2. Software simulation approach
The software simulation platforms for Smart Grid are entirely composed of software components, which can also be further classified into two subcategories: individual simulation platforms and co-simulation platforms.

2.2.1. Individual simulation platforms: Individual simulation platforms are those which encapsulate the simulation features into one process. In other words, it is one single simulator to complete the job. These platforms are usually focused on a particular aspect of interests for Smart Grid. In [Guo et al. 2008], Guo et al. implement an energy demand management simulator to predict the performance and response of a self- adaptive demand management strategy. In [Molderink et al. 2009], Molderink et al. design and develop a simulation environment from scratch to analyze control algorithms for various appliances, such as micro-generators, energy buffers and water heater, etc. In [Faria et al. 2010], Faria et al. describe Demsi, a simulator for demand response in the context of competitive electricity markets and intensive use of distributed generation. Energy service provider and demand side player are modeled and strategic decisions are evaluated. In [Narayan 2008], Narayan et al. propose GridSpice, a cloud-based simulation package for Smart Grid. Employing the well known distribution network simulator Gridlab-D [Chassin et al. 2008] and the transmission network simulator Matpower [Ray D. Zimmerman 1999], GridSpice is being developed iteratively with an ultimate goal of modeling the interactions between all parts of the electrical network, including generation, transmission, distribution, storage and loads. All the individual software platforms can complete their own tasks in the specific application domain, but they all just concentrate on the power network simulation. The communication network, as another critical component of Smart Grid, is not considered in these platforms. This is why the co-simulation platform comes to the picture.

2.2.2. Co-simulation platforms: Co-simulation (co-operative simulation) is a simulation methodology that allows individual components to be simulated by different simulation tools running simultaneously and exchanging information in a collaborative manner [Li et al. 2014]. In [Hopkinson et al. 2006], Hopkinson et al. present a federated simulation combining NS2, a discrete event network simulator with PSCAD, a continuous time power network simulator. In [Godfrey et al. 2010], Godfrey et al. simulate the Smart Grid using NS2 and OpenDSS, a power network simulator. In [Mallouhi et al. 2011], Mallouhi et al. introduce a co-simulation testbed specifically for security
analysis of SCADA system by employing PowerWorld simulator and OPNET. In [Lin et al. 2011] and [Deng et al. 2013], Lin et al. introduces a global event queue to synchronize NS2 and PSLF simulation.

The co-simulation approach typically requires iteratively running separate communication and power network simulations. The performance is affected by putting extra overhead of an intermediary of synchronization. Meanwhile, the interactions between communication and power system models are usually restricted to fixed synchronization interval. Mismatches can occur between the real dynamics and the simulated one, which exposes reliability issues of such systems. An improvement about this issue is to integrate one simulation component into the other, such that a single global clock and event queue is employed in the simulation engine. In [Mets et al. 2011], electric network is made into a component within OMNET++, a network simulator. In [Nutaro et al. 2007], the adevs simulation tools are integrated into NS2 to provide a hybrid modeling of the continuous time power system and discrete event communication system by the discretization of the continuous power dynamics. More recently, a few of co-simulation frameworks are developed to further improve the interoperability between multiple individual simulation platforms. [Ciraci et al. 2014] introduce FNCS, a co-simulation framework to incorporate multiple power system simulators (Matpower, GridLAB-D) and communication network simulator (NS3). [Lehnhoff et al. 2015] present Mosaik, which allows the Smart Grid users to combine thousands of simulated entities distributed over multiple simulator processes.

2.3. Remarks about related work

From the above literature review, we summarize the characteristics of the real hardware testbed approach and the software simulation approach for experiments in Smart Grid.

The real hardware testbed approach achieves high fidelity by employing dedicated devices as part of the platforms. Critical control programs, such as demand response algorithms, routing protocols, and security strategies, can be tested in real hardware testbeds and they could be directly migrated to the actual Smart Grid embedded devices. However, the substantial cost and resource needed to deploy these devices limits the repeatability of these efforts in a lab environment. Moreover, these testbeds cannot be accessed and shared remotely by the public research community and are difficult to scale when the test case becomes quite large.

The software simulations, on the other hand, achieve much better availability, usability and scalability. Within software simulation platforms, the models of various Smart Grid objects can be easily scaled and statistically analyzed. However, since software simulation typically abstracts the operating system, communication protocols and power dynamics into various mathematical simulation models, it can only duplicate the behavior and structure of the system, but not the execution environment of Smart Grid applications. Moreover, many of the recent smart grid cases are hard to model because either they are in binary executable forms (e.g. malware codes), or evolve too rapidly (attack vectors), which makes the simulation development labor-intensive and error-prone.

2.4. Our approach: Software-Hardware Hybrid and Federated Experiment Environment

In light of the above issues, a much ideal approach is to combine the merits from both hardware and software platforms such that the users may examine the performance of Smart Grid applications under realistic communication and computation constraints in hardware platform, while evaluating the corresponding scalability in software platform at the same time.
ScorePlus bridges the gaps between real hardware approach and software simulation approach. The key advantages of ScorePlus are:

— First, ScorePlus employs both software emulator and hardware testbed, which expose the same and transparent interface to users. A Smart Grid application can be tested on either of them without any modification. With this integration, researchers may examine the performance of Smart Grid applications under realistic communication and computation constraints in hardware testbed, while evaluating the corresponding scalability in software emulator at the same time.

— Second, the federated architecture of ScorePlus enables each distributed software emulator or hardware testbed maintain its own autonomy and unique strengths, while all work together to make their resources available under a unified framework. This plug-and-play architecture greatly facilitates the scalability of distributed experiments.

— Finally, as far as we know, ScorePlus is the first Smart Grid experiment tool that has close integration within cloud infrastructure. Leveraging the customized resource plugin mechanism of OpenStack cloud computing platform, the ScorePlus software are equipped with high reusability to support massive deployment with large scale test cases.

ScorePlus also has its own limits: the power network model is static DC power flow model such that we cannot use ScorePlus to capture transient physical dynamics, frequency control, power balance, and voltage regulations, etc. The strengths and limitations of our approach compared with related works are listed in Table I. The ScorePlus codes are open source released at https://sourceforge.net/projects/scorepluset/.

<table>
<thead>
<tr>
<th></th>
<th>Real hardware testbed</th>
<th>Software Simulation</th>
<th>ScorePlus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model fidelity</td>
<td>High</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Accessibility</td>
<td>Difficult</td>
<td>Easy</td>
<td>Easy</td>
</tr>
<tr>
<td>Scalability</td>
<td>Low</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Code migration</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Time step</td>
<td>Real time</td>
<td>Real time/discrete time</td>
<td>Real time</td>
</tr>
<tr>
<td>Frequency control, Power balance</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Voltage regulations</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Cloud Infrastructure integration</td>
<td>N/A</td>
<td>N/A</td>
<td>Yes</td>
</tr>
</tbody>
</table>

3. Overall System Design

Figure 1 demonstrates the overall architecture of our platform. ScorePlus consists of Graphical User Interface (GUI), software emulator, and hardware testbed. The GUI connects with software emulator and hardware testbed remotely through Internet. Since the software emulator and the hardware testbed expose the same architecture and interfaces to the GUI, the user can run the same Smart Grid application test case on either of them without any migration issue. Meanwhile, by integrating the communication network and power network from both software emulator and hardware testbed, multiple software emulators and hardware testbeds are able to connect and interact through Internet. Upon specifying the connection interfaces between each other, communication and power flow could be established between each individual node, such as a virtual demander in software emulator and a real supplier in hardware testbed.
4. Software Emulator

As shown in Figure 2, the software emulator consists of Service Layer, Virtual nodes, Linux Ethernet Bridging, Communication Module, and Power Module. The Service Layer is essentially a socket server that provides various event handlers to the formatted messages from external interacting system. It is responsible for initializing the emulation case, collecting and forwarding the system request, and managing multiple emulation sessions, etc. The software emulator partially leverages our previous development in [Tan et al. 2012]. In this work, we specifically implement the network tunnels between virtual network interfaces and physical network interfaces, and upgrade the power network model through domain decompositions, and increase more message handles in the Service Layer to respond to external interactions, etc. All the improvements in the software emulator are intended to facilitate its integration with the hardware testbed.

4.1. Virtual Nodes: Light Weighted Virtualization

The emulation feature of software emulator is implemented using Linux network namespace techniques, which is a recent light weighted paravirtualization technique supported by mainstream Linux kernel. By calling the clone() system call, each created virtual node can have its own instance of Linux OS network stack and process space while sharing the same local file systems and hardware with other virtual nodes. From the perspective of codes running inside, each virtual node is just another piece of hardware platform controlled by Linux OS. Therefore, the virtual nodes can directly execute unmodified Smart Grid application codes from physical Linux-embedded devices, and vice versa. Figure 3 illustrates the software emulator scalability that about 180 virtual nodes can be created on a 64 bits HP desktop with Pentium(R) Dual-Core CPU E5700 @ 3.00GHz and 4GiB memory.
4.2. Linux Ethernet Bridging and Communication Module

All the virtual nodes in the emulator are equipped with virtual Ethernet interfaces and they are linked by Linux Ethernet Bridging. This approach provides underlying communication capabilities between each virtual node. Based on Linux Ethernet Bridging, the communication module in software emulator provides comprehensive support of various wired and wireless communication network models. All the models are simply the manipulations of the underlying communication infrastructure. Each emulated node has its own instance of OS implemented TCP/IP stack supported by Linux namespaces, from the perspective of Open Systems Interconnection (OSI) model, thus it allows high fidelity emulation of network layer and above. By default, a simplified simulation of link and physical layers is enabled, which is implemented using netem with Ethernet bridging in Linux. Statistical network effects such as bandwidth, loss rate, bit error rate and noise level can be configured and applied. For higher-fidelity link and physical layer emulation, other network simulation tools, such as EMANE [Ahrenholz et al. 2011], can be directly integrated. More importantly, virtual Ethernet interfaces can be directly mapped to a physical Ethernet interface on the emulation host, such that all the traffic passing through that physical port can be transferred to the emulation environment. This enables the interactions between the software emulation environment with outside physical networks.
We employ the above key features to achieve the integration of communication networks between the software emulator and hardware testbed, which will be presented in Section V.

4.3. Power Module

The power module in software emulator emulates the power flows analysis within Smart Grid and also provides implementations of pre-defined energy models. The power module receives initial power network topology, energy model configuration information and the connection interface information from service layer to formulate the power network model.

4.3.1. Power Network Model: General Description

The power network model is a static DC circuit model to emulate power flow analysis. Assume a power grid is composed of \( n \) nodes and \( b \) branches. Since the power network dynamics is subject to Kirchhoff’s current and voltage laws (KCL and KVL), in order to calculate the voltages of all nodes, we apply nodal analysis to the grid and get the linear equations for the whole system:
\[
AV = I
\] (1)

where coefficient matrix \( A \) is the \((n - 1) \times (n - 1)\) reduced nodal admittance matrix since we have chosen a reference node. Let \( Nbr(i) \) represents the neighbour set of node \( i \) in the power network, we get:
\[
a_{ij} = \begin{cases} 
\sum_{s \in Nbr(i)} g_{is} & i = j. \\
-g_{ij} & j \in Nbr(i) \\
0 & otherwise 
\end{cases} \] (2)

\( g_{ij} \) is the admittance between node \( i \) and node \( j \), \( V \) and \( I \) are the unknown node voltage vector and the known nodal current injection vector, respectively. Figure 4 shows the data flow diagram of power module. The power module receives initial power network topology and energy model configuration information from service layer to formulate the initial power network model. The model is updated when the corresponding model updates are received from the interactions with communication module. Meanwhile, incremental model updating is adopted to react to the system status changes. More importantly, as the size and order of the power network increase, a single power network model can be distributed into multiple computing instances, as further shown in section 4.3.3.

4.3.2. Incremental updating

Based on previous model, let’s consider the situation when the power network topology changes. Suppose the power grid status changes, such as the admittance between node \( i \) and node \( j \) is changed by \( \Delta g_{ij} \). So the new coefficient matrix \( \tilde{A} \) can be written as:
\[
\tilde{A} = A + U \Delta g_{ij} U^T
\] (3)

where
\[
U = \begin{bmatrix} 0 & \cdots & 1 & \cdots & -1 & \cdots & 0 \\ i & j & & & & & \\ \end{bmatrix}^T
\]

Particularly, the changed admittance \( \Delta g_{ij} \) equals to \( -g_{ij} \) when the branch is removed and \( \Delta g_{ij} = g_{ij} \) when a new branch is added. Notice that [Zhang et al. 2007]
\[
\tilde{A}^{-1} = A^{-1} - A^{-1} U (\Delta g_{ij}^{-1} + U^T A^{-1} U)^{-1} U^T A^{-1}
\] (4)
then we can get the $\tilde{A}^{-1}$ with a much lower computation cost when we store previously computed $A^{-1}$.

4.3.3. Power Network Model: Domain Decomposition Power network is generally a network of loosely coupled sub power networks. Each sub network is a relatively independent partition of the whole energy system and only few in-between connection lines join them together. Inside each sub network, we divide the nodes into two sets:

- Internal nodes: nodes that only have connections with the nodes inside the same sub network.
- Boundary nodes: nodes that have connections with the nodes in other sub networks.

The architecture of the power network is illustrated in Figure 5. Based on the previous analysis, we apply the Schur complement domain decomposition method [Saad 2003] to our power network model. Specifically, suppose there are $k$ sub networks, by grouping the internal nodes of each sub network and putting all the boundary nodes of the network in the back, we formulate the nodal analysis model for the whole power network.
network as the following:
\[
\begin{bmatrix}
Y_{A_1} & 0 & \cdots & 0 & Y_{A_1B} \\
0 & Y_{A_2} & \cdots & 0 & Y_{A_2B} \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & \cdots & Y_{A_k} & Y_{A_kB} \\
Y_{B_1A} & Y_{B_1A} & \cdots & Y_{B_kA} & Y_{B_kA} \\
\end{bmatrix}
\begin{bmatrix}
V_{A_1} \\
V_{A_2} \\
\vdots \\
V_{A_k} \\
V_{B_1} \\
\end{bmatrix}
= 
\begin{bmatrix}
I_{A_1} \\
I_{A_2} \\
\vdots \\
I_{A_k} \\
I_B \\
\end{bmatrix}
\] (5)

Notice that $B$ is the set of all boundary nodes in the whole network, consisting of $B_1, B_2, \ldots, B_k$. Therefore, $Y_{A_iB}$ only has non zero entries in its submatrix $Y_{A_iB_i}$, for all $i = 1, 2, \ldots, k$.

From (5), if the voltages for boundary nodes set $V_B$ is known, then the voltages for the nodes in each sub network can be calculated as the following:
\[
Y_{A_i}V_{A_i} = I_{A_i} - Y_{A_iB}V_B, \forall i \in \{1, 2, \ldots, k\}.
\] (6)

Meanwhile, if we keep the corresponding part for the boundary node set $B$ in equation (5), we can get:
\[
\tilde{Y}_{BB}V_B = \tilde{I}_B
\] (7)

where
\[
\tilde{Y}_{BB} = Y_{BB} - \sum_{i=1}^{k} Y_{BA_i}Y_{A_i}^{-1}Y_{A_iB}
\] (8)
\[
\tilde{I}_B = I_B - \sum_{i=1}^{k} Y_{BA_i}Y_{A_i}^{-1}I_{A_i}
\] (9)

Define
\[
x_i = Y_{BA_i}Y_{A_i}^{-1}Y_{A_iB}
\] (10)
\[
y_i = Y_{BA_i}Y_{A_i}^{-1}I_{A_i}
\] (11)

for all $i = 1, 2, \ldots, k$. Notice that $x_i$ and $y_i$ only requires local information for sub system $i$.

We employ the above key feature to achieve the integration of power networks between the software emulator and the hardware testbed, which treats each of them as a sub network of the whole power network. The details for that would be given in section V.

5. Hardware Testbed

Figure 6 shows the design of our hardware testbed, which follows the same architecture as the software emulator. Each node in the hardware testbed is a physical energy device emulating an energy entity in Smart Grid system. Through communication interface and power interface, all the energy devices are connected by communication network and power network.

5.1. Overview of Energy Devices

The hardware testbed is composed of the following energy devices to emulate the energy entities in a Smart Grid system:

— 1 Supplier (This device emulates a general power generation. Output to the Smart Grid is up to 200mA. The Smart Grid voltage is typically 2.5V, but it may be higher or lower when source and load are unbalanced.)
5 Solar Panel Controller (Output is up to 30mA each, depending on light intensity.)
5 Wind Turbine Controller (Output is up to 30mA each, depending on wind intensity.)
5 Storage (Capacitor bank with electronics. It can source or sink up to 50mA depending on the demands of the Smart Grid system.)
15 Demander (A load only, drawing up to 20mA each from the Smart Grid.)
5 Topology Switch (Has 6 ports and can switch current flow in multiple ways. Must be able to handle up to 300mA on all ports.)
1 Interface device, which serves as the energy tunnel when the hardware testbed is connected with the software emulator.

Figure 7 shows various energy devices, solar panel and wind turbine in use. The LCD display shows the current drawn and sourced by each energy device except the one in Topology Switch Device, which displays the connection status between all the 6 ports.

5.2. Energy Device Design Details

Each of the above energy device includes three boards: a Beagleboard [Beagleboard 2009], a Telosw board [Lu et al. 2010] and an Energy board. Figure 8 shows the process of remote access and configuration of these energy devices from GUI by the user. The user specify the hardware configurations from the GUI and send requests to the Service Layer as formatted messages. Upon receiving and mapping these messages, the Service Layer forwards the corresponding request to the Beagleboard of the designated energy device. Then the control program in that Beagleboard then will communicate with its Telosw board, which ultimately interacts with the Energy board and put those configurations into effect, such as load of the demander, output of the renewable controller and connection status of the topology switch, etc. Likewise, the status...
of the energy devices are periodically queried and reported to GUI backward. The roles of the above three boards are summarized as the following:

![Diagram of energy device communication](image)

**Fig. 8. Remote access and configuration of energy devices**

5.2.1. **Beagleboard**

- Interact with the Service Layer and control the behavior of each energy device correspondingly.
- Provide Linux-based environment in each energy device to test the Smart Grid applications.
- Enable the communications between each energy device, through both wired and wireless networks.

5.2.2. **TelosW board**

- Adjust of smart resistors to emulate power profiles.
- Monitor/measure power generation and consumption rate of each energy device.
- Provide I/O to mating Energy board for LED and display.

5.2.3. **Energy board**

- Provide different DC current circuit to fulfill the corresponding power requirements of different devices.

![Diagram of solar panel controller](image)

**Fig. 9. Details of Solar Panel Controller**

Figure 9 shows the architecture of a Solar Panel Controller. The Beagleboard provides physical communication interfaces, such as the Ethernet port and mini USB
to connect the wireless radio. It is connected with TelosW board through USB cable such that the programs running in the Beagleboard can directly access and control the energy profile through the TelosW. The TelosW is mated to the Energy board, which are all enclosed in a black plastic container. The port at the top of the container is connected with a physical solar panel. There are three ports at the bottom of the container. The left one serves as the power source of the device and the right one is the actually power interface for Smart Grid system. The middle port is connected to the Ground.

Figure 10 shows the schematics for the Energy board on Solar Panel Controller. An analog voltage from the TelosW board indicates the amount of current output to the Smart Grid that is desired (0 to 2.0V corresponds to 0 to 30mA of desired output). An analog voltage to the TelosW board indicates the measured current output (0 to 2.0V corresponds to 0 to 30 mA of actual output current). The Solar Cell input to the Energy Control board will have a capacitor of 10,000uF (C6) for the purpose of temporarily storing energy from the generating source. 0.4 times the voltage on this capacitor will be supplied to the TelosW board for the purpose of indicating energy capability of the Solar Cell. The same device can be used for wind turbine control. Figure 11 shows a sample current output of a Solar Panel Controller device when the solar panel is exposed to a 50W 3-level light intensity lamp over time. Figure 12a shows the relationship between voltage input via current output in Controller device. Figure 12b shows the screenshot of the ADC output voltage of the Solar Panel Controller in oscilloscope. The voltage output is very stable and no oscillation occurs.
5.3. Power Network in Hardware Testbed: Dynamic Topology Configuration

The power network in hardware testbed connects all the energy devices through power interfaces through power cables. To facilitate the dynamic configurations of various power network topologies, we design the Topology Switch device as Figure (7d), which serves as the current hub for all the other energy devices. The Topology Switch can be accessed and configured dynamically through TelosW by the programs running inside the mated Beagleboard, such that the connection status between the 6 ports can be changed accordingly. Different power network topologies can be set up based on the users’ requirements. The connection status is also visualized through the LCD. The Energy board of Topology Switch is shown in Figure 13.

5.4. Communication Network in Hardware Testbed: Wire and Wireless Network

The communication network in hardware testbed are set up to emulate the wire and wireless communication in Smart Grid. The Generic Routing Encapsulation (GRE) tunnels are created between each energy device above the underlying communication network, such that the physical communication network can be unified with the ones in software emulator. For wired network, we use Ethernet to connect each energy device.

For wireless network, we are employing the WISP-2 outdoor antenna from ALFA Network, Inc, which can be directly connected with Beagleboard through USB cable.
WISP-2 is a Low-Cost IEEE 802.11n outdoor AP/CPE operating in 2.4GHz band that is also compliant with the standard IEEE 802.11b/g. In our indoor environment, the radio range of each node covers all the other nodes, so physically, all nodes can communicate with each other. To formulate a wireless mesh network with a specific topology, the iptables tool chain is employed within each energy device. Through ipables, we can filter the sent and received packets for each node, such that they can only communicate with the nodes as specified. Then the corresponding wireless network topology is formulated. The performance testing result from iperf tool are listed in Table II.

<table>
<thead>
<tr>
<th>hops</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>bandwidth</td>
<td>1.05 Mbps</td>
<td>624 Kbps</td>
<td>416 Kbps</td>
<td>316 Kbps</td>
<td>242 Kbps</td>
</tr>
<tr>
<td>jitter</td>
<td>1.83ms</td>
<td>37.14ms</td>
<td>55.91ms</td>
<td>83.26ms</td>
<td>122.55ms</td>
</tr>
</tbody>
</table>

6. Integrating Software Emulators and Hardware Testbeds

ScorePlus supports scalable distributed experiments such that multiple software emulators and hardware testbeds running at different (local or remote) locations are able to connect and form a larger Smart Grid system. Here we only present the integration between software emulator and hardware testbed. Note integrations only among software emulators or only among hardware testbeds follow the same mechanism. In order to enable the interactions between the software emulator and the hardware testbed, both the communication network and power network within them should be integrated.

6.1. Integrating the Communication Network

The communication network emulated in software emulator runs in real time, so they can be connected to live physical networks. We build GRE tunnels between/among software emulation servers and the energy devices in hardware testbed. Figure 14 illustrates the details of how a virtual node in software emulator sends a packet to a physical node in hardware testbed. GRE tunneling is built between the software emulation server and the gateway machine of the hardware testbed. The actual IP address of the two are in domain 131.96.x.0/24. When the packet reaches the edge of the network in software emulator, which is the physical network interface of emulation server, the tunnel entry in routing table would enable the encapsulation of the packet with GRE header and the tunnel destination, such that they can reach the hardware testbed environment. Also, by using this approach, all the ip addresses in the experiment environment are in 10.0.x.x/16 domain, and the physical networks can be abstracted as needed for the experiments.

![Fig. 14. Communication between virtual node and real node through GRE tunneling](image)
Figure 15 presents the performance of integrated communication network by showing the time elapsed for Open Shortest Path First (OSPF) routing protocol to converge. We employ different numbers of hardware testbed nodes and record time under different network delays set up in software emulator. As shown in Figure 15, when hardware testbed node is integrated within the communication network of the software emulator (the number of real nodes from 0 to 2), there is a dramatic increase in elapsed time. This indicates the GRE tunneling does impose a significant overhead. When the network delay in software emulator is increased to be comparable to the one in hardware testbed, which is 10ms this case, we see that the number of hardware testbed nodes involved in experiment doesn’t really affect much the convergence speed of routing. The virtual and physical network gives about the same performance.

6.2. Integrating the Power Network

The power network integration employs the domain decomposition model in Figure 5. Instances from either hardware testbeds or software emulators are treated as sub networks of whole power network.

Specifically, suppose there are already $k$ connected instances of software emulators and hardware testbeds, and another instance $k + 1$ joins in run time. Also assume that the $k + 1$ instance connects with the instance set $E$ directly, $E \subseteq \{1, 2, ..., k\}$. Then the composition process for each computation host $i, i = 1, 2, ..., k, k + 1$, is executed as the following:

- If $i \in E$, then adjust the boundary node set $B_i$ by adding the new boundary nodes connected with instance $k + 1$ and also adjust the internal node set $A_i$ by removing the corresponding boundary nodes connected with instance $k + 1$. Compute $x_i$ and $y_i$ based on equations (10) and (11) respectively. Send the results to the coordinator host.
- The coordinator first reforms the boundary node set $B$ by adding the new boundary nodes in $B_i, i \in E$ and $B_{k+1}$, then rebuilt $Y_{BB}$ and $I_B$ for the whole system. Secondly, it collects $x_i$ and $y_i$ from each host, and calculate $V_B$ based on equations (7) (8) (9). Finally, it sends $Y_{BB}V_B$ back to each host.
- Each host $i$ receives $Y_{BB}V_B$ from the coordinator and calculate $V_{A_i}$ based on equation (6).

After the above process, each sub power network can set its updated status based on the calculated results. If the sub network is a software emulator, we can update the status in our programs directly to achieve the resulting effects. However, if the sub network is a hardware testbed, physical changes must be made. To this end, we employ our specifically designed Interface device, which is essentially a combined supplier and demander with large capacity. When the calculated result indicates the hardware testbed sub network is requesting power from outside domains, the Interface device
is set as a power supplier to provide corresponding power to the testbed. When the calculated result indicates the hardware testbed sub network is providing power to outside domains, the Interface device is set as a power demander to absorb power from the testbed correspondingly.

When one sub network is disconnected with the rest of the system, the steps are similar with the above except that instead of adding boundary nodes to the boundary set, the coordinator will remove the boundary nodes related with the exiting instances.

7. Deployment Plugin for ScorePlus in OpenStack Cloud Computing Platform

ScorePlus employs Linux containers (LXC) to achieve virtualizations in software emulator. As shown in Figure 3, a general PC can support test cases with at most 180 virtual nodes. As the scale of the test case grows, it becomes indispensable to deploy the software emulator in cloud computing infrastructure to increase its scalability. Moreover, when setting up multiple software emulation instances with different configurations, it would be convenient to just specify different parameters somehow without starting from scratch each time. To facilitate these deployment processes, we particularly implement a deployment plugin for ScorePlus in OpenStack cloud computing platform, which is essentially a resource plugin in Heat, the OpenStack Orchestration service.

7.1. OpenStack and Heat

OpenStack is a free and open-source cloud computing software platform, which is a global collaboration of developers and cloud computing technologists to produce the open standards for both public and private clouds. The OpenStack software consists of a group of interrelated projects to control various aspects of cloud infrastructure, such as authentication, orchestration, computing, networking, and storage etc.

Heat is the main project in OpenStack Orchestration program, which provides a template based orchestration to describe a cloud application. A Heat template allows instantiations of various OpenStack resource types, such as instances, floating ips, volumes, security groups, and users, etc. All the resources allocated for a Heat template are managed as a single stack. More importantly, Heat allows developers to add customized resource plugins, such that new resources can be integrated into the OpenStack frameworks to create cloud applications.

7.2. Implementation of Heat Plugin for ScorePlus

We design and implement a customized Heat resource plugin for ScorePlus in OpenStack, such that ScorePlus can be deployed and reused the same as any other resources in OpenStack cloud applications. In this way, ScorePlus can run on different computing, networking, and storage resources with different configurations, which are all based on the specifications in a Heat template. The resource plugin extends a base Resource class, and the lifecycle is managed by a series of relevant handler methods, including create, update, suspend, resume and delete. Figure 16 illustrates the state transition diagram of ScorePlus resource plugin within Heat engine. Specifically, the following key methods extended from the heat.engine.resource.Resource class are implemented to handle the entire lifecycle:

- handle_create: Deploy ScorePlus on the dependent server and start the ScorePlus service socket.
- handle_update: Update current ScorePlus instance.
- handle_suspend: Pause the ScorePlus service socket process.
- handle_resume: Resume the ScorePlus service socket process.
- handle_delete: Remove ScorePlus related process and software on the dependent server.
7.3. Sample Use

Figure 17 shows a sample Heat stack template using ScorePlus resource plugin, which also includes other resources like OS::Nova::Server, OS::Heat::SoftwareConfig and OS::Heat::MultipartMime, etc. Figure 18 captures the corresponding console output after the deployment of this template in OpenStack. With respect to the dependencies between different resources, we can also see the dependency topology for the allocated resources from this template stack in Figure 19.
8. Evaluation and Experimentation

In this section, we demonstrate and evaluate the capabilities of ScorePlus to support cyber-physical analysis in Smart Grid, particularly in Microgrid [Hartono et al. 2013]. The future Smart Grid is expected to be an integration of Microgrids featured by localized power generation, storage and consumption [Liang et al. 2012]. Microgrid works as an independent local power system that has the flexibility to connect (connecting mode) and disconnect (islanding mode) from the main grid as needed in order to minimize the energy cost and maximize the grid stability [R. Kamel and Nagasaka]. Figure 20 illustrates a typical Microgrid structure. Note that in ScorePlus we use Topology Switch to serve as circuit breakers.

8.1. Experiment setup

The test case is created upon the AMI network test case in [Sarfi et al. 2012] and demand response model in [Hammerstrom, D. J. 2007]. The demand response model is also run in @GridLAB-D as a comparison for accuracy. We investigate the Microgrid demand response behavior in one virtual day. For each virtual time period $h$ in the virtual day (2 real time seconds), $h = 0, 1, 2, 3, ...23$, each kind of nodes behaves as the following:
Supplier: Providing power with maximum 2kw to the whole grid and broadcasting its hourly price [Hammerstrom, D. J. 2007] to all the topology switches. Calculating the bills for each demander based on the real time price and the collected energy consumption from Demanders.

Topology Switch: Receiving the energy price information from the Supplier and then relay the messages to the demanders immediately.

Solar Panel: When $h \in [0, 5] \cup [19, 23]$, setting its maximum output to 0. When $h \in [6 : 18]$, randomly setting its maximum output between 200w and 300w.

Wind Turbine: Randomly setting its maximum output between 100w and 200w for each hour.

Storage: Its capacity is set to 1kwh with initial energy 0.1kwh; SENDING its current energy residual to demanders and charges/discharges with maximum 100w as requested from Demanders.

Demander: Either critical Demander or adjustable Demander. Critical Demander must work at 200w all the time. Adjustable Demander receives price from Topology switch and energy residual from Storage and adjust its setpoint based on the demand response model in [Hammerstrom, D. J. 2007].

An AMI communication network is set up between all the nodes, which consists of both Ethernet and IEEE 802.11 RF Mesh network with 54Mbps bandwidth, 0.1% packet loss rate. All the above node behaviors are implemented by the control programs running inside each node. Figure 21 shows Xterminal for a Demander node. We use command `ps aux` to check programs running inside. They include the OSPF routing process and the demander energy daemon process.

Five Microgrids are created as in Table II, running on four software emulation servers and one hardware testbed respectively.

8.2. Islanding Mode

In islanding mode, each Microgrid is working independently and self-sustained. We first examine Microgrid 1 and 2 for comparison. Figure 22 shows the real-time energy...
price and the adjustable Demander energy consumption over time, one in software emulator, one in hardware testbed and one in GridLAB-D as comparison for accuracy. We can see that the control program for the price responsive model in [Hammerstrom, D. J. 2007] tends to shift the energy consumption to the lower price period of the day. In addition, we can see the difference of the power profiles between the two adjustable Demanders. Theoretically, the two curves should be exactly the same since they are controlled by the same program as previously specified. However, since the software emulator is event-based processing, it can only update with a large discrete step size. As a comparison, due to the high ADC sampling rate of TelosW, the one in hardware testbed has the advantage of capturing the transient dynamics in a finer grain.

Figure 23 shows the renewable share of total energy consumption over time in all five Microgrids. We can see that generally, when the real time energy price increases, the renewable share increases correspondingly. In addition, as the size of the Microgrid increases, the renewable share become less sensitive to the price fluctuations since more renewable energy surplus can be distributed and stored by the demand response process.
8.3. Connecting Mode

In connecting mode, Microgrid is connected with each other. We first examine our implementation for the integrations between different Microgrids. The software emulation servers and hardware testbed are connected with each other through Internet. Note that we specifically design the five Microgrids in Table II, such that Microgrid 1 connected with 2 is the same as Microgrid 3, and Microgrid 3 with 4 is the same as Microgrid 5. The study case No. and the Microgrids connected are listed in Table III.

<table>
<thead>
<tr>
<th>Case No.</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microgrid connected</td>
<td>1,2</td>
<td>3</td>
<td>2,3</td>
<td>4</td>
<td>3,4</td>
<td>5</td>
<td>1,2,4</td>
<td>1,3,4,5</td>
</tr>
</tbody>
</table>

Figure 24(a) shows the wind turbine output in case 1 and 2. The two curves conform with each other and the integration between software emulator with hardware testbed preserves the power profiles accurately. Figure 24(b) shows the ping time between two farthest nodes and the time needed for OSPF routing to convergence in case 3, 4, 5, 6, 7. It indicates that for communication network, the integrations with software emulators introduce minor delays but major delays with hardware testbed, which is about 0.57 ms difference in ping time.

8.3.1. Deployment in OpenStack For the multiple ScorePlus instantiations in case 8, we employ the ScorePlus resource plugin to deploy them all at one time in OpenStack. In particular, Neutron networking resources are integrated into the stack template, such that a private internal subnet is created to connect all 5 ScorePlus servers, all of which uses a common gateway router to communicate with external networks. The result networking topology of the multiple ScorePlus servers in OpenStack are illustrated in Figure 25.
8.3.2. **Cyber-Physical Attacks** In connecting mode, the influence between different Microgrids becomes extremely important since the failure in one Microgrid could impact the stability of other Microgrids. To this end, we conduct contingency analysis for case 1 under topology attacks, in which attackers intend to disrupt the normal physical conditions of power system through malicious control of Topology Switch. As in Figure 26, suppose attackers inject a malicious control program to the Topology Switch n4 in house 3 and runs it at time h=7, such that it disconnects house 4 with house 5 as well as the hardware testbed. Only port 2, 3 and 4 are connected. In normal condition, the Topology Switch in house 3 is set such that port 1, 2, 3, 4, 5, 6 are all connected. So the power needed in house 3 can be supplied from house 4, house 5 and the power network hardware testbed. After manipulating the connection status in the Topology Switch, the previous balanced power flow in the network is changed to unbalanced one, leading to a dramatic increase in the power flows (indicated by the red bold line in GUI). Meanwhile, since the hardware testbed is not providing power to the software emulator any more but the power generation is not aware of the attack at the moment, the extra power flow would be inevitably forwarded to the Storage devices in hardware testbed, which results in a sudden peak charging rate at the storage device, as marked by the red circle. The charging rate will reach a sudden peak around h=7 but return a higher but relative stable level afterwards since more energy surplus is being forwarded to Storage device. The same experiment is also conducted by replacing the hardware testbed with another software emulator. We can see that the hardware testbed could capture the power dynamics in a finer grain.

9. **Conclusion**

In this paper, we present the design, implementation and evaluation of ScorePlus, a software-hardware hybrid and federated experiment environment for Smart Grid. Our platform provides an extendable design paradigm for the creation of general cyber-physical testbeds. Testing cases such as distributed control algorithms, demand responses, and cyber-physical security issues, can all be evaluated in our platform. Future work would investigate the design of more hardware components like Phaser Measurement Unit and the decrease in system integration overhead. Moreover, third party evaluation will be conducted to validate the performance more thoroughly and management mechanisms for open access will be created. Also, other wireless communication technologies will also be explored to integrate within hardware testbed. The ScorePlus codes including both software emulator and hardware testbed are open source released at https://sourceforge.net/projects/scorepluset/.
REFERENCES


Reza Barzegaran. 2015. Attributes and design of Resilient Renewable Microgrid Laboratory. 10th CMU workshop for smart grid testbed (2015).


Boming Zhang, Shouchen Sun, and Zheng Yan. 2007. Advance Power analysis.