DSSnet: A Smart Grid Modeling Platform Combining Electrical Power Distribution System Simulation and Software Defined Networking Emulation

Christopher Hannon
Illinois Institute of Technology
10 West 31st Street
Chicago, Illinois, 60616
channon@hawk.iit.edu

Jiaqi Yan
Illinois Institute of Technology
10 West 31st Street
Chicago, Illinois, 60616
jyan31@hawk.iit.edu

Dong Jin
Illinois Institute of Technology
10 West 31st Street
Chicago, Illinois, 60616
dong.jin@iit.edu

ABSTRACT

The successful operations of modern power grids are highly dependent on a reliable and efficient underlying communication network. Researchers and utilities have started to explore the opportunities and challenges of applying the emerging software-defined networking (SDN) technology to enhance efficiency and resilience of the Smart Grid. This trend calls for a simulation-based platform that provides sufficient flexibility and controllability for evaluating network application designs, and facilitating the transitions from in-house research ideas to real productions. In this paper, we present DSSnet, a hybrid testing platform that combines a power distribution system simulator with an SDN emulator to support high fidelity analysis of communication network applications and their impacts on the power systems. Our contributions lay in the design of a virtual time system with the tight controllability on the execution of the emulation system, i.e., pausing and resuming any specified container processes in the perception of their own virtual clocks, with little overhead scaling to 500 emulated hosts with an average of 70 ms overhead; and also lay in the efficient synchronization of the two sub-systems based on the virtual time. We evaluate the system performance of DSSnet, and also demonstrate the usability through a case study by evaluating a load shifting algorithm.

Keywords

Electrical Power System Simulation; Network Emulation; Software-Defined Networking; Smart Grid; Microgrid

1. INTRODUCTION

Today's utilities increasingly adopt modern communication network technologies to realize their Smart Grid initiatives. For example, the growing development of "smart microgrid", a core component of the future integrated smart gird, is highly dependent on the successful operation of the underlying communication networks. A microgrid focuses on the power distribution system that can operate independently or in conjunction with the traditional main power grid. The microgrid approach focuses on creating a plan for local energy delivery that meets the exact needs of the constituents being served, and introduces huge economic and environmental benefits to our society. For example, the IIT campus microgrid [3] has achieved a 6.58% reduction in annual CO2 emission (saving 3,457,818 kg), and a unit price of 7 cents per kilowatt-hour, while the average price in the U.S. is 10.43 cents per kilowatt-hour in 2015 [2].

To build a resilient and secure networking environment for microgrids and other smart grid applications, we and other researchers envision a software-defined networking (SDN) enabled network infrastructure for the critical power control systems [11,15,17,22,25]. Figure 1 depicts our design of the next-generation IIT campus microgrid. SDN offers the global network visibility, which would enable detailed virtualization and facilitates network and traffic management. With direct and centralized network control integrated with the existing grid control application, we now allow more intelligent utility applications to blossom, such as system-wide configuration verification and context-aware detection systems.

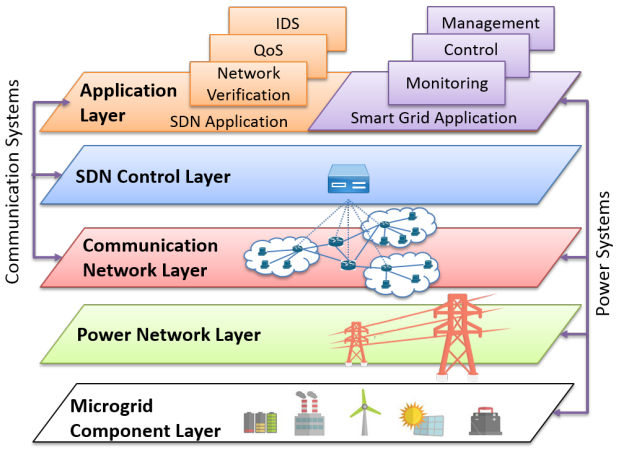


Figure 1: A Multi-Layered SDN-enabled Microgrid Design

However, incorporation of new technologies in such critical control systems is very challenging, because of strong real-time requirements, continuous system availability and many

SIGSIM-PADS '16, May 15–18, 2016, Banff, AB, Canada.

DOI: http://dx.doi.org/10.1145/2901378.2901383

resource-constrained legacy devices. Therefore, a testing platform targeting such cyber-physical systems is strongly needed for the research community to evaluate the new network technologies and their impact on the power grid systems, before the real deployment. In this paper we present **D**istribution **S**ystem **S**olver **Net**work (DSSnet), a hybrid simulation-emulation testbed incorporating an electrical power distribution system simulator and an SDN-based communication network emulator, with the following features. First, DSSnet enables the modeling of a modern power distribution system and simulates the Intelligent Electrical Devices (IEDs) that make it up. Second, DSSnet enables high fidelity analysis by allowing real networking applications to run in the network emulator and interact with the power simulator. Third, DDSnet provides flexible and direct network programmability by supporting real SDN switch and controller software, an inherent advantage by adopting Mininet [19].

A key challenge is synchronizing the execution of the power simulator and the container-based emulator. This is because all the processes in the emulator execute real programs and use the system clock to advance experiments, while the simulator executes models to advance experiments with respect to its simulation virtual clock. To address this issue, we refine a prior virtual time system [26] and develop a new capability to enable pausing and unpausing the emulation container processes by modifying the Linux kernel. Our underlying design shows how the challenge of synchronizing time and events between the two systems is possible using virtual time, while ensuring high fidelity. We perform extensive evaluation of the system, including system overhead and experiment fidelity in terms of network flow throughput and latency. In addition, we demonstrate the usability of DSSnet with a case study on analyzing the effectiveness of a load shifting algorithm and evaluating the power system impact under a denial-of-service attack.

In the remainder of the paper, Section 2 presents the related work and shows the differences of DSSnet with those existing tools. Section 3 describes the system design and how it addresses the synchronization challenges across two systems. Section 4 presents the component-level implementation. Section 5 evaluates the system performance. Section 6 demonstrate a load shifting application that illustrates the features and benefits of DSSnet. Finally, Section 7 concludes the paper with future works.

2. MOTIVATION AND RELATED WORK

2.1 Combining Power with Communication

The power grid is composed of power generation, transmission, distribution and loads. Traditionally, power is generated in mass quantities from hydro, coal, nuclear, and gas sources. The power is then transmitted at high voltages to distribution systems where the power is distributed to residential and commercial consumers. As the power grid is moving towards a smarter grid, the efficient energy management is increasingly dependent on the underlying communication network supporting reliable information transfer among the various entities in the grid.

With distributed power generation—such as solar and wind energy—and more storage technology, there is a need for understanding the state of the power network in real time. A challenge with the integration of such generation, is the uncertainty and intermittency of the availability of power generation. In order to combat this challenge, there needs to be an infrastructure that allows for the monitoring and control of the system state. To do this effectively, requires a reliable and resilient communication network.

Researchers have developed systems to co-simulate the power and network components of the smart grid [9, 10, 12, 14, 16, 20, 23]. [21] surveys the existing technologies and motivations for co-simulation.

In [23], a system is proposed using OpenDSS to allow for sending real-time signals to hardware integrating with simulation. Real time simulators are used for hardware-in-the-loop simulations, allowing for simulation-emulation closer to the real system [12]. This gives high fidelity, but requires power equipment and often specific simulator hardware. Using a network emulator we make the system closer to that of real hardware deployment, but without the cost or complexity associated with real hardware.

In [20], the authors create a co-simulation between PSLF and ns-2. They use a global event driven mechanism for sending synchronization messages between the two simulators. In simulation, events are sorted by time stamps, typically in a priority queue. To enforce temporal order of events, we take inspiration from the global event queue, and adapt this strategy to integrate the network emulation with the distributed power simulation in DSSnet.

EPOCHS [16] uses commercial power simulators to co-simulate network and power systems through the use of agents. This platform uses agents to effectively co-simulate power and communication elements. The authors define agents as having the properties of autonomy and interaction. That they exhibit properties of mobility, intelligence, adaptivity and communication. In DSSnet, our models run real processes in the network emulation. This allows for us to make use of agents to as entities that exist in both systems.

FNCS [10] is a federated approach for co-simulation of power and electrical simulators by combining multiple power simulators, both distribution and transmission and use ns-3 as a communication simulator. In [9], the same authors improve the synchronization between systems that we take inspiration from in our implementation in Section 4. The difference is that DSSnet is focused on network emulation which has different synchronization challenges due to the inherent difference between the execution mechanisms in simulation and emulation.

There are two main features that set our design apart from the existing tools. The first is that we are using a network emulator rather than a simulator. The emulator allows for higher fidelity by executing real networking programs. The second is that our network emulator supports SDN-based networks.

2.2 Software Defined Networking in Utility

Software defined networking (SDN) is an emerging network technology that separates the data plane from the control plane. The benefit of this is the enhanced ability to have a global view over the network and be able to program network switches to provide functions that were previously too laborious and impossible to do. SDN allows for complex network functions to be created by adjusting network paths and flows in real time — reactively and proactively. This technology can help solve security issues and increase per-

formance in many networks such as data centers, and even in energy infrastructure. However SDN is not widely used yet and does not solve all problems out of the box.

In [17], SDN is proposed to allow for scalable deployment of utility applications. The authors show how SDN can provide network functions to simplify publisher-subscriber roles in intelligent electrical devices (IED) including in phasor measurement unit networks.

In [11], the authors propose a system that combines an SDN emulator with an off-the-shelf high voltage solver. The difference between the system they propose and ours is that we are focused on combining open source tools and that our simulator is for low voltage distribution networks.

In [15], SDN is utilized to increase the performance of SCADA networks. In our testbed we have also modeled SCADA network elements, which can be used to explore how cyber attacks can impact the power grid using different communication models.

In [25] the authors analyze utility communication networks for situational awareness including during blackouts. Through the use of a hybrid power and communication system, situational awareness can be enhanced to increase the resilience of the grid.

Additionally, there has been work to bring existing power grid network protocols such as GOOSE and IEC 61850 into SDN networks [22]. Our testbed can be used to emulate IEC 61850 based communication with the advantage of analyzing the effects in the power simulator.

To summarize, our system is built on top of a network emulator rather than the existing works of network simulation for high fidelity analysis in the context of smart grid, and the emulator we use supports SDN-enabled software switches and protocols.

3. SYSTEM DESIGN

DSSnet integrates a distribution power system simulator, OpenDSS [6], with a network emulator, Mininet [19], using virtual time. The system has the following features:

- Power Flow Studies
- SDN-based Communication Network Modeling
- Smart Grid Control Applications
- Virtual-Time-Enabled Network Emulation

DSSnet is composed of five main components: the communication network emulator, the electrical power simulator, a network coordinator for interfacing with the network and the virtual time system, a power coordinator for interfacing and controlling the simulator, and a virtual time system which manages time and ensures synchronization in DSSnet. Figure 2 depicts the architecture of DSSnet.

3.1 System Design Architecture

3.1.1 Network Emulator

The network emulator in DSSnet contains software switches that emulate the function of real SDN switches. In DSSnet, the hosts represent IEDs in a power network, and each host has its own virtual network ports. Hosts in the emulation have their own namespaces [19] and can run real processes to model IEDs. Any element in the power network that has a communication requirement can be modeled in the emulation, including SCADA elements such as sensors and phasor measurement units (PMU), and even relays and generators. Load management devices are presented in both systems, such as smart loads and smart meters.

Another benefit to having each model run its own process(es) is that not all network processes need to be present in the simulator. In the network, some hosts interact with the simulator indirectly through other models, such as data collection and storage systems, state estimation applications, voltage and frequency adjustment controllers.

There are drawbacks to using emulation. With each host running its own processes and having their own virtual network adapter, the system becomes more complex, making debugging a challenge. Most importantly, emulation cannot scale to sizes as large as thousands and hundreds of thousands oh hosts like simulation can due to virtualizing hosts which requires many resources. Our future work includes the development of distributed emulation to achieve better scalability, with reference to a prior work on the distributed OpenVZ-based network emulator [28].

3.1.2 Power Simulator

DSSnet models define the power network through elements such as lines, transformers, relays, meters (sensors), loads, capacitors, and generators. Each IEDs behavior in the power simulator can be modeled in the network emulator. However, not all power elements need to be represented in the network emulation, since some elements may exist only in the power network. The power simulator begins by initiating all of the power elements and creating an element matrix representing how the elements affect each other over time. The purpose of the power simulation is to simulate the behavior of utility distribution systems. Functions of the power simulator include power flow snapshot, harmonic study, fault studies, load modeling, and solving dynamic time step power flow [13].

In simulation, the amount of execution time required to solve for the state at a given simulation time step depends on the nature of the request. Typically, small time steps at the level of microseconds may be required for protection studies, while larger time steps at the level of milliseconds may be required for power flow studies while load and generation studies may be at large scales such as seconds, and minutes.

3.1.3 Network Coordinator

The network coordinator starts the network emulation, and creates data structures to maintain a centralized view of the network. The network topology and the IED models are loaded through the coordinator to configure network properties. The role of the network coordinator is also to interface with the communication network emulator for synchronization between the communication and power systems. The coordinator listens for synchronization event requests from the hosts through the event queues. When the coordinator receives a synchronization request, it interfaces with the power coordinator and with the virtual time system to control DSSnet's virtual clock.

3.1.4 Power Coordinator

The power coordinator interfaces with the power simulator. The first task of the module is to initiate the power simulator by setting up the circuit options and initially solving the circuit in a snapshot at time 0. The power coordinator provides an API to modify and the extract values in

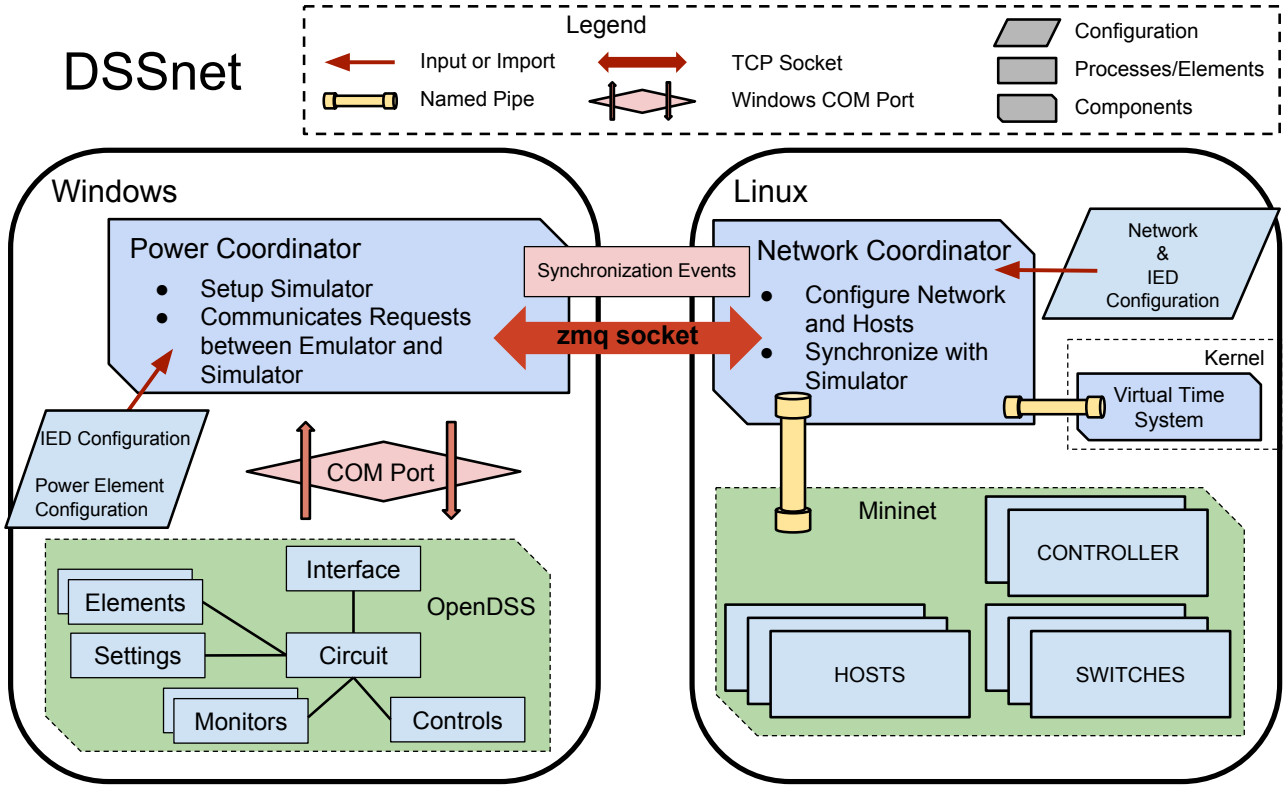


Figure 2: DSSnet system architecture diagram. Note that the power simulator runs on a Windows machine and the network emulator runs on a Linux machine.

the simulator. The simulator is able to accept synchronization events from the network coordinator through the API and return a response accordingly. A role of the module is to advance the simulation's clock to the time stamp of the current event request and to solve the power flow at that time. Additionally, some elements of the power grid may be modeled in the power coordinator as a function of time, such as loads and generation. These elements are not necessarily represented in the communication network, but can still operate on DSSnet's virtual clock.

3.1.5 *Virtual Time System*

Unlike simulation, the emulation clock elapses with the real wall clock. Therefore, pausing the emulation requires more than just stopping the execution of the emulated entities, but also pausing their clocks. Virtual time can be used to achieve this goal [18, 26]. We choose to extend the work of [27], in which Mininet is patched with virtual time support. However, their motivation is different from ours.

In general, virtual time has at least two categories of application. The first one is to slow down emulation so that it appears to emulated entities that they have sufficient *virtual* resources. Slowing down execution also alleviates the problems caused by resource multiplexing. The work in [27] and [26] fall into this category. Another usage of virtual time is for emulation-simulation synchronization. In DSSnet, we assign every container a private clock, instead of using the global time provided by the Linux OS. The containers now have the flexibility to slow down, speed up or stop its own clock when synchronizing with the simulator.

However, the emulator needs to manage the consistency across all containers. This is achieved by a centralized timekeeper in [18], and by a two-layer consistency mechanism in [26]. A more flexible virtual time system implemented by [26] avoids this problem as emulation takes charge of this responsibility. In practice, the emulator configuration guarantees that all containers are running with one shared virtual clock; Similarly, the container leverages the Linux process hierarchy to guarantee that all the applications inside the container are using the same virtual clock. The two-layer consistency approach is well-suited to this work for pausing and resuming because:

1. All hosts should be paused or resumed when we stop or restart the emulation.
2. All processes inside a container should be paused or resumed when we stop or restart the emulation.

The first task is done by the network coordinator. The second task is implemented based on the fact that processes inside a container belong to the same process group.

3.2 Synchronization

A key challenge in DSSnet is the synchronization between connecting the emulated communication network and the simulated power system. The root cause is that two different clock systems are used to advance experiments. Ordinary virtual-machine-based network emulators use the system clock, and a simulator often uses its own virtual clock. This difference would lead to causality errors as shown in the following example.

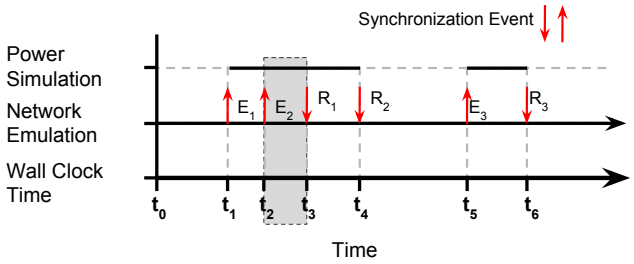


Figure 3: The execution of DSSnet is shown with respect to the wall clock. The network emulation runs concurrently with the power simulation, and is not paused which allows for synchronization errors to occur, when requests arrive before the responses are sent, e.g., R_1 occurs after E_2. The shaded box highlights the location of the error.

In Figure 3, there are three cross-system events (E_i), each with a response (R_i). E_1 occurs before E_2, however, E_2 may require information from R_1. Since the response occurs after the second event, the global causality is violated, and thus reduces experiment fidelity. An example of E_1 is a request to retrieve power flow values while E_2 sets the value of a discharging battery based on the value returned previously. Since the reply R_1 occurs after E_2 this can introduce an error. Furthermore, such errors can be accumulated if the simulation keeps out of synchronization with the emulation.

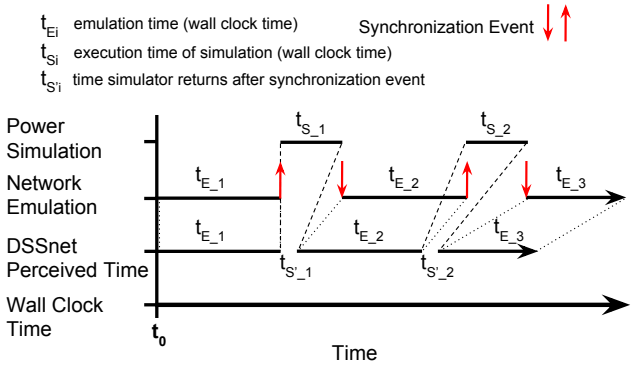


Figure 4: The execution of DSSnet is shown with respect to its own perceived time, i.e., the sum of the emulation execution time (can be dilated or not dilated) and the virtual time elapsed in simulation. The network emulation is paused to allow for the simulation to catch up to the emulation—this also ensures synchronization errors in the early example do not occur.

To address this issue, we develop a virtual time system in Mininet with the new capability to pause the emulator without advancing the emulation virtual clock, while the simulator is running. We adopt this idea, since the experiment advancement in DSSnet by design is driven by the emulation. Before the coordinators permit the simulator to advance over a time interval [a,b), we first ensure that all processes in the emulator have advanced their own clocks to at least time b, to ensure that all input traffic that arrives at the simulator with timestamps in [a,b) are obtained first.

Figure 4 shows the execution of the DSSnet. The total execution time (equation 1) is the total time the emulation is running plus the sum of the time spent executing the simulation. DSSnet's clock (equation 2) is equal to the total time of the emulation plus the sum of the returned simulation virtual times. In this illustrative scenario, we do not include factors like synchronization overhead, parallel execution based on simulation and (possibly) emulation lookahead, and time dilation effect in emulation virtual time, for simplicity.

$$Time_{wall\_clock} = \sum t_{E\_i} + \sum t_{S\_i} \quad (1)$$

$$Time_{DSSnet} = \sum t_{E\_i} + \sum t_{S'\_i} \quad (2)$$

$$ret = \frac{t_{S'\_i}}{t_{S\_i}} \quad (3)$$

where ret's value range is

- (1,∞) if the power simulation takes longer time to execute than the real time; Thus, emulation virtual time is essential for synchronizing the two systems
- (0,1] if the power simulation takes less or equal time to execute than the real time, i.e., with real-time simulation capability
- 0 if the power simulation time is not considered by the emulation; for instance, recomputing voltage and current change along power lines at nearly light speed.

Synchronization events occur when either system influences the other. One optimization is to divide the global queue into two queues, because synchronization events can be created in two ways: Non-Blocking Events and Blocking Events. For each type of event, we design a queue sorted by time stamps to organize the requests. The non-blocking event queue contains premeditated synchronization events and events that do not require a response to the communication network. For example, the non-blocking event queue can be used to pass messages to the simulation to sample the power flows with meters at periodic intervals. Other examples are power events such as line faults that occur at a specific time.The IEDs are able to influence the power simulation by sending a synchronization event message using the blocking queue. Examples of these classes of synchronization events are that PMUs requesting values from the power simulation and controllable loads changing power values or turning on or off.

By using the non-blocking event queue, we can speed up the overall execution time. In other words, we do not need to pause the emulation for non-blocking events (E_1 and E_3 in Figure 5). However, if a blocking synchronization event (E_2 in Figure 5) occurs before the response R_1, then the emulation is paused at t_2, i.e., the time stamp of E_2. The emulation is resumed at t_4, when response R_2 is returned. In this work, we demonstrate the advantage of having a non-blocking queue with sample events. How to classify the events is not a focus for this paper. In addition, the container-based emulation system introduces opportunities for offering real application specific lookaheads to improve the parallelism performance, which we will explore as our future work.

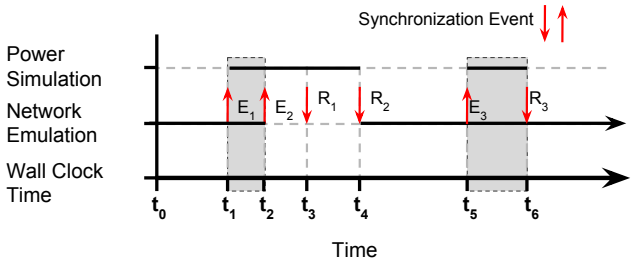


Figure 5: E_1, and E_3 are non-blocking synchronization events and E_2 is a blocking synchronization event from an IED. The network emulation is not paused unless an event in the blocking queue occurs, i.e., the one that requires a response to the communication network. The shaded box represents the portion of the experiment that is running in parallel.

4. IMPLEMENTATION

DSSnet combines Mininet, an SDN emulator, and OpenDSS, an electrical power distribution system solver simulator. This section presents implementation details with our algorithmic contributions.

4.1 Network Coordinator

The network coordinator is implemented as a python program running on the Linux machine. The network coordinator is responsible for (1) initializing the experiment with inputs like network topology and IED configuration, and (2) interfacing with the processes running on Mininet, the virtual time system, and the power coordinator, using named pipes. The network coordinator listens on a named pipe for synchronization calls, and also opens a connection to the power coordinator using ZeroMQ library [7] for python. It sends synchronization event requests and handles the reply from the power simulator.

4.2 Power Coordinator

The power coordinator is implemented as a python program running on the Windows machine. It directly controls OpenDSS through the provided COM port. Because the IED agents may exist in both the power and network systems, the power coordinator maintains a mapping between OpenDSS elements and Mininet hosts for the associated IEDs. The power coordinator listens on a TCP/IP port for a request from the network coordinator. After receiving the request, the message is parsed and handled according to user defined functions. The user has the ability to interface with the provided APIs, such as `set_time` and `solve`, but also can implement direct commands or query custom values from OpenDSS, such as PMU value requests, and setting load values. Part of the request specifies if a reply is required to the network coordinator and if so, the power coordinator sends a reply based on the user-defined handler.

4.3 Virtual Time System for Network Emulation

We extended a prior work on virtual-time-enabled Mininet [27], and implement the emulation pausing and resuming capability. To do that, we develop two routines *freeze* and *unfreeze* in the Linux kernel.

4.3.1 Freeze/Unfreeze Interface

The virtual file system provides an interface between the kernel and the user space. Since virtual time is a per-process property, it is more efficient to create a `/proc` file entry for the associated processes rather than adding system calls. The virtual time interface consists of two extra file entries under `/proc/$pid`.

- `/proc/$pid/dilation` A process `$pid` can enable and disable virtual time, as well as change a new time dilation factor (TDF). To support fractional dilation values, a TDF of x is stored in this entry as $1000x$, since floating point numbers are rarely supported in the Linux kernel.
- `/proc/$pid/freeze` We can freeze and unfreeze a process `$pid` according to the written boolean value. A value 1 freezes the entire process group and a value 0 resumes all the processes in this group.

We make a distinction between regular processes and virtual-time enabled processes. In other words, the `/proc/$pid/freeze` entry is only valid only if `/proc/$pid/dilation` already has a non-zero value. The emulator can enable a container with virtual time by writing 1000 to the `dilation` proc file entry. This will turn on the freeze/unfreeze capability without unnecessarily modifying the clock speed. In this work, we use a process calling system call `unshare()` with flag `CLONE_NEWTIME` to enable virtual time. This design is motivated and tailored to be compatible with Mininet's programming interface.

We also develop a user space utility program `freeze_all_proc`. This program can freeze and unfreeze multiple hosts in parallel. In particular, it spawns one `pthread` for every network host to write its freeze entry in the Proc system. Since the network coordinator always pauses or resumes all hosts, this optimization significantly reduces the running overhead in large-scale network settings.

4.3.2 Freeze/Unfreeze Implementation

To track virtual time using the OS software clock, we add several new fields into the process descriptor `task_struct`.

- `dilation` represents the time dilation factor of a time-dilated process. We also use `dilation` as a flag to indicate whether a process virtual-time-enabled or not.
- `physical_start_ns` represents the starting time that a process detaches from the system clock and begins to use the virtual time, in nanoseconds.
- `physical_past_ns` represents the amount of elapsed physical time since the last time inquiry, in nanoseconds.
- `freeze_start_ns` represents the starting time that a process or a process group is frozen. It is always zero for a non-frozen process.
- `freeze_past_ns` represents the cumulative time, in nanoseconds, that a running process or a process group remains in the frozen state.

Algorithm 1 shows the procedure to enable, disable and update virtual time. The for-loop (line 33) cascades the update TDF operation to all child processes. The algorithm

Algorithm 1 Set Time Dilation Factor

```
1: function INIT_VIRTUAL_TIME(tsk, tdf)
2:     if tdf > 0 then
3:         __getnstimeofday(ts)
4:         now ←timespec_to_ns(ts)
5:         tsk.virtual_start_ns ← now
6:         tsk.physical_start_ns ← now
7:         tsk.dilation ← tdf
8:     end if
9: end function
10:
11: function CLEANUP_VIRTUAL_TIME(tsk)
12:     tsk.dilation ← 0
13:     tsk.physical_start_ns ← 0
14:     tsk.physical_past_ns ← 0
15:     tsk.freeze_start_ns ← 0
16:     tsk.freeze_past_ns ← 0
17: end function
18:
19: function SET_DILATION(tsk, new_tdf)
20:     old_tdf ← tsk.dilation
21:     vsn ← tsk.virtual_start_ns
22:     if new_tdf = old_tdf then
23:         return 0
24:     else if old_tdf = 0 then
25:         INIT_VIRTUAL_TIME(tsk, new_tdf)
26:     else if new_tdf = 0 then
27:         CLEANUP_VIRTUAL_TIME(tsk)
28:     else if new_tdf > 0 then
29:         OLD_DILATION_TIMEKEEPING(tsk, new_tdf)
30:     else
31:         return -EINVAL
32:     end if
33:     for all child of tsk do
34:         SET_DILATION(child)
35:     end for
36: end function
```

to freeze/unfreeze processes is shown in Algorithm 2, and is implemented in the Linux kernel. After stopping a group of processes, we record the current time for calculating the process frozen duration once we unfreeze the process. Note that sending `SIGCONT` to all processes is behind the time keeping function. The reason is that if we resume the process group first, an unfrozen process may be scheduled to run, and possibly query time before we complete populating the `freeze_past_ns` within the entire container.

5. SYSTEM EVALUATION

5.1 Virtual Time System Overhead in Network Emulation

As described in Section 3, the synchronization between the power simulator and the network emulator requires us to freeze and unfreeze all emulated hosts. These operations bring overhead to synchronization. The overhead is not tolerable when the scale of the networking system grows to hundreds of emulated hosts on a single physical machine, which is quite common in practice [19]. Note that the overhead to freeze/unfreeze processes does not affect the emulation temporal fidelity, which is evaluated in the next section.

Algorithm 2 Freeze and Unfreeze Process

```
1: function FREEZE(tsk)
2:     kill_pgrp(task_pgrp(tsk), SIGSTOP, 1)
3:     __getnstimeofday(&ts)   /* timespec ts */
4:     now ← timespec_to_ns(ts)
5:     tsk.freeze_start_ns ← now
6: end function
7:
8: function POPULATE_FROZEN_TIME(tsk)
9:     for all child of tsk do
10:        child.freeze_past_nsec ← tsk.freeze_past_nsec
11:        POPULATE_FROZEN_TIME(child)
12:    end for
13: end function
14:
15: function UNFREEZE(tsk)
16:    __getnstimeofday(&ts)   /* timespec ts */
17:    now ← timespec_to_ns(ts)
18:    tsk.freeze_past_ns+ = now − tsk.freeze_start_ns
19:    tsk.freeze_start_ns ← 0
20:    POPULATE_FROZEN_TIME(tsk)
21:    kill_pgrp(task_pgrp(tsk), SIGCONT, 1)
22: end function
```

We measured the overhead of our `pthread`-based implementation by repetitively freezing and unfreezing emulated hosts. We varied the number of hosts as 10, 50, 100, 250, 500 in Mininet. For each setting, we repeated the freezing and unfreezing operations for 1000 times, and computed the overhead as the duration from the moment the coordinator issues a freezing/unfreezing operation to the moment that **all** hosts are actually frozen/unfrozen. We added the overhead of freezing operation and the overhead of the associated unfreezing operation, and plotted the CDF of the emulation overhead in Figure 6.

We observe that more than 90% of the operations take less than 100 milliseconds in the 500-host case. For all other cases, more than 80% of the operations consume less than 50 milliseconds. We also observe the average overhead time grows linearly as the number of hosts increases in Figure 7. The error bars indicate the standard deviations of the overhead time, which are caused by the uncertainty of delivering and handling the pending `SIGSTOP` and `SIGCONT` signals.

5.2 Accuracy Evaluation

End-to-end throughput and latency are two important network flow characteristics. In this section, we use these two metrics to evaluate the communication network fidelity. We created two emulated hosts connected via an Open vSwitch in Mininet. The links are set to 800 Mbps bandwidth and 10 μs latency. `iperf` [4] was used to measure the throughput, and `ping` [5] was used to measure the round-trip-time (RTT) between the two hosts.

5.2.1 End-to-end Flow Throughput

We used `iperf` to transfer data over a TCP connection for 30 seconds for throughput testing. In the first run, we advanced the experiments without freezing the hosts. In the second run, we froze the emulation for 1 second, and repeated the operation every 1 second for 64 times during the data transmission. We coupled the two experimental results and reported the average throughputs between the

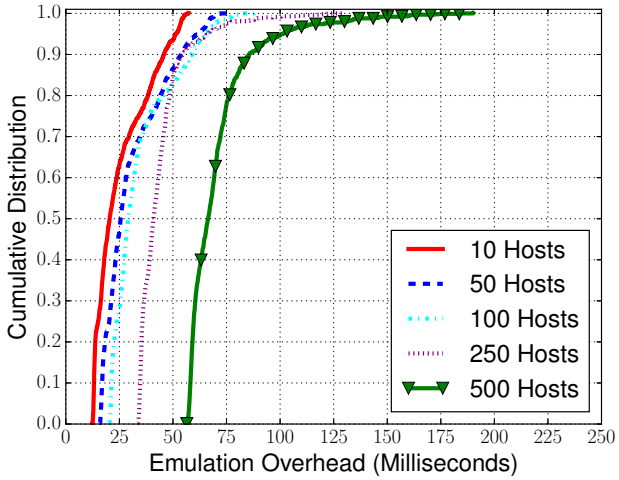


Figure 6: CDFs of Network Emulation Overhead Caused by Freezing/Unfreezing Operations

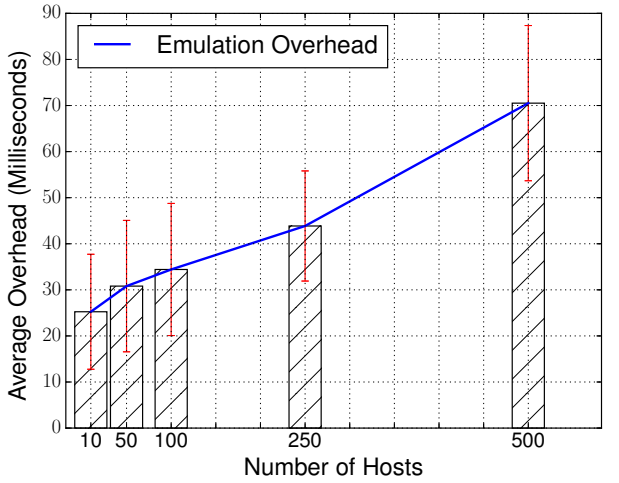


Figure 7: Average Network Emulation Overhead

11th second and the 30th second in Figure 8. The error bars represent the 99% confidence interval of the throughputs.

We observed that the average throughputs of the "interrupted" emulation matches well with the baseline results. However, pausing emulation introduces around 11% – 18% deviation. Several sources lead to this deviation. First, while we explicitly generate `SIGSTOP` and `SIGCONT` signals to the containers, those signals are only in the pending state. The actual deliveries depend on the OS scheduler, and the deliveries usually occur when exiting from the interrupt handling. Second, the actual freezing duration depends on the accuracy of the sleep system call. Sleeping for one second has a derivation about 5.027 milliseconds on the testing machine, Dell XPS 8700 with Intel Core i7-4790 3.60 GHZ processor.

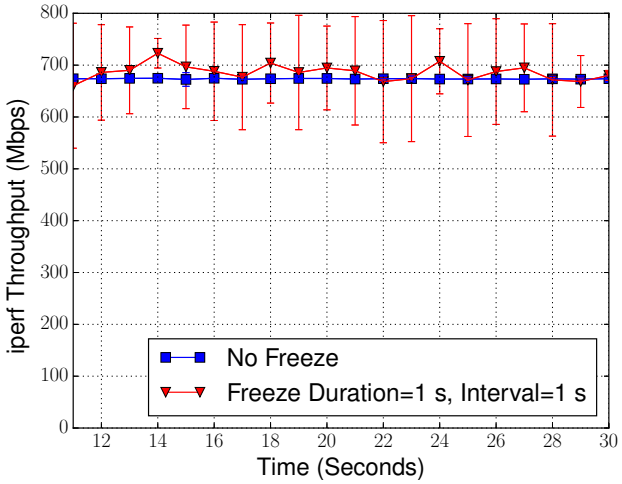


Figure 8: TCP Flow Throughput Comparison, 800 Mbps Bandwidth and 10 μs Link Latency

5.2.2 *End-to-end Flow Latency*

To evaluate the end-to-end flow latency, we issued 1000 `ping`s with and without freezing the emulator. We skipped the first ping in the results to exclude the effect of ARP and the switch rule installation from the SDN controller. Figure 9 plots the CDF of the round trip time (RTT) for both sets of `ping` experiment. We observed the two lines are well matched in the case of 10 μs link delay, and pausing the emulator does not affect the distribution of RTT. About 80% ping packets are received around 0.2 ms.

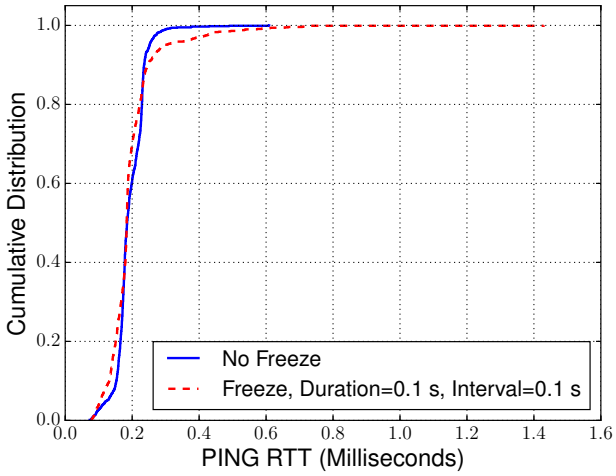


Figure 9: Ping Round-Trip-Time Comparison, 800 Mbps Bandwidth and 10 μs Link Latency

When we increased the link latency to 1 millisecond, the observed RTTs in the freezing emulation case were around 1 ms slower than the non-freezing case. One solution is to reprogram the `hrtimer`, but if the target kernel only supports low resolution timers, we need to search in the complicated time-wheel structure, otherwise we can search in a red-black

tree. Another approach is to explore the emulation lookahead to increase the synchronization window size, and thus reduce the synchronization frequency between the two systems. We will leave those enhancements as our future work.

6. CASE STUDY: LOAD SHIFTING

DSSnet is designed to be used for smart grid applications that affect both the power grid and the communication network. We now present a case study of analyzing the load shifting problem using DSSnet.

6.1 Load Shifting Problem

We consider a class of loads called "shiftable loads". Shiftable loads are power consuming elements including Hybrid Power Electric Vehicles (HPEV)s, appliances such as dishwashers, pool pumps, Heating and Air Conditioning (HVAC), water heaters, refrigerators, etc. Some of them are preemptive, such as car, pool, HVAC, and others are non-preemptive, such as washing machine, dryer. The demand pattern can be represented with a peak during the day, near 4 pm and an absolute minimum near 4 am [1]. This load shifting problem is surveyed in [24], and a mix integer linear programming algorithm proposed in [8] where the authors considered more constraints on non-preemptive loads. For simplicity we only look at preemptive ones and ignore possible local minima and maxima costs over time. In this case study, we use a novel greedy polynomial time approximate algorithm proposed as follows.

6.1.1 Problem Formulation

Given the following definitions

N_L : number of loads

N_{TS} : number of time slots

L_i : i^{th} load

TS_j : j^{th} time slot

S_i : start time of i^{th} load

F_i : finish time of i^{th} load

c_i : rated power (cost) of i^{th} load

h_i : number of time slots required

P_{ij} : power consumed by i^{th} load at time slot j

V_j : maximum power at j^{th} time slot

Q_i : scheduled or full time steps eligible for i^{th} load

f_j : forecasted price at j^{th} time slot

The problem is to minimize the total cost of power

$$Cost_{total} = \sum_{j=1}^{N_{TS}} \sum_{i=1}^{N_L} P_{ij} * f_j \tag{4}$$

The following constraints must be satisfied for load requirements.

- Consumption Constraint declares that loads only consume power and do not produce power:

$$\forall (i,j) : P_{ij} \geq 0 \tag{5}$$

Algorithm 3 Greedy Load Shifting Scheduler

```
1: Let L be the set of all loads
2: Let TS be the set of all time slots
3: function SCHEDULE(L_i)
4:     for TS_j ∈ TS do    /* O(N_TS) */
5:         if TS_j.sched[L_i.id] is TRUE then
6:             Continue
7:         else if L_i.power > TS_j.volume then
8:             Continue
9:         else if TS_j.time ∈ [S_i, F_i) then
10:            TS_j.sched[L_i.id] ← TRUE
11:            TS_j.volume = TS_j.volume − L_i.power
12:            Break
13:        end if
14:    end for
15:    RECALCULATE_SCHEDULABILITIES(L, TS)
16:    BUILD_HEAP(L)    /* O(N_L) */
17: end function
```

- Temporal Power Constraint declares that every load must consume its full power between its start and end time:

$$\begin{cases} \forall i : \sum_{S_i}^{F_i} P_{ij} = h_i * c_i \\ \forall i : \sum_{j=1}^{S_i} P_{ij} = 0 \\ \forall i : \sum_{j=F_i}^{TS_{N_{TS}}} P_{ij} = 0 \end{cases} \tag{6}$$

- Volume Constraint declares that the maximum amount of power available at each time slot cannot be exceeded:

$$\forall j : V_j \geq \sum_{i=1}^{N_L} P_{ij} \tag{7}$$

6.1.2 Scheduling Algorithm

In order to schedule the loads, we define the schedualability factor p in equation 8:

$$p_i = \frac{h_i}{F_i - S_i - Q_i} \tag{8}$$

It is the ratio of the number of time periods are required for a load over the number of time slots are available for that load. We maintain all load items in a heap based on this value, and sort time slots by price. We select the load at the top of the heap (the hardest load to schedule), and check if we can add it to the first time slot. Algorithm 3 shows the steps within each iteration. After successful scheduling, we update all load elements schedulability p. The 'slots available' property of the load is decremented by 1, if the new volume of the time slot is smaller than the rated power of the load, and if Q_i has increased. Next we recreate the heap. If any load has a p value less than 1 then the algorithm fails. However, for sufficiently large number of time slots and volumes, this is unlikely. Algorithm 3 will be iterated maximally $N_{TS} * N_L$ times, because each element only has less than N_{TS} time slots. For-loop search takes maximum $O(N_{TS})$ time. Rebuilding the heap after each successful schedule is $O(N_L)$. The total running time of our algorithm is thus $O((N_{TS} + N_L) * N_{TS} N_L)$.

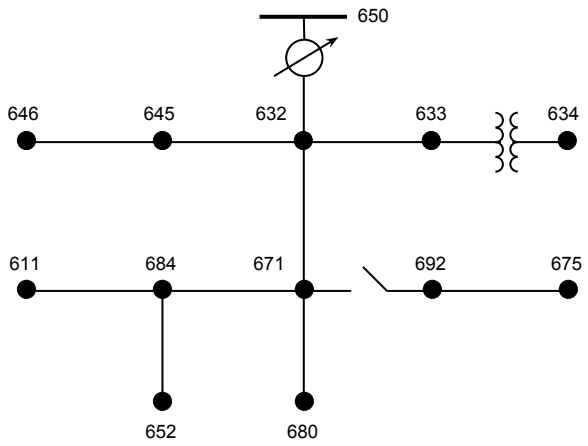


Figure 10: 13-Bus Distribution System

6.2 Experiment Setup

The data to be used for this algorithm consists of loads, prices, and time slots. The load data was modeled from the average daily charge for Electric Vehicles(EVs), and the start time and end time were modeled from the distribution of when people return to their homes in the evening. Other loads were considered that have a shorter duration and varying start time to model appliances. In total, 130 load categories were created, each representing 10 loads. The price used is from real ComEd historic hourly data [1]. The price points between the hours were calculated linearly.

We started our day at the forecasted maximum demand and ran for 6 hours. The time slots were each 5-minute long and spanned a total of 6 hours, generating 72 time slots. The load and time slot data generated represent a time period from 6 pm to 12 am, where people are returning home and plugging in their cars and using appliances. The start times and end times for each load falls within this window.

In this experiment, base loads were arranged according to the specifications of the IEEE 13 bus reference circuit. Shiftable loads replaced the loads connected to buses 611, 652, 680, and 675 in Figure 10. The base load is a linear decreasing value, power equals to $5313 - \frac{3}{20} * t$ with a small amount of noise ($\pm$20kW) added and where t means time in seconds since 6:00 pm. At bus 650, we measured the power entering the distribution system.

Figure 10 depicts the simulated power network. The communication network has one SDN switch at each bus, connected along the power lines. All the links in the communication network are 10 Mb/s links. The coordinator starts Mininet, the load models, and the power application scheduler on the hosts.

The load shifting algorithm runs as a real-time scheduler residing as a power application host in DSSnet. The scheduler is connected to the switch at the substation, which communicates in real time with the loads—hosts in DSSnet—through TCP/IP communication. The performance is evaluated by measuring the state variables within the power simulator. The impact on the power grid are determined by monitoring the power flow into the distribution network at bus 650.

In this setup, the power application scheduler acts as the server and the loads act as clients. The server will send load updates *on* or off and the loads will then send the updated value as a synchronization event. During a synchronization event, the power coordinator updates the load variable in the simulator, advances the simulators clock to the time of the emulation and solves the power flow problem. The simulator also samples a monitor at the infinite bus 650 and exports the data in a log file. Because there is no return value injected from the power simulator back to the emulator, these events are sent as non-blocking resulting in a faster overall run time. Both the loads and scheduler are modeled as real processes. To the best of our knowledge, DSSnet is the only smart grid testbed that allows for this kind of interaction between processes.

The importance of using DSSnet to evaluate the smart grid application is to see the effect on the power grid when the communication network experiences changes. What sets DSSnet apart from related works is that in DSSnet, real attack mechanisms can be used rather than just simulating the effects. In this case study, we consider a denial of service attack (DoS). We present a DoS attack at t=7:30, in which the power application server goes offline. The DoS attack can be accomplished by flooding the server with TCP requests and denying any other hosts the ability to connect. Because the load models require communication from the load scheduling server, if the communication is down, they revert back to the default schedule. In this experiment, all communication is blocked from the loads to the scheduler after 90 minutes to emulate a DoS attack.

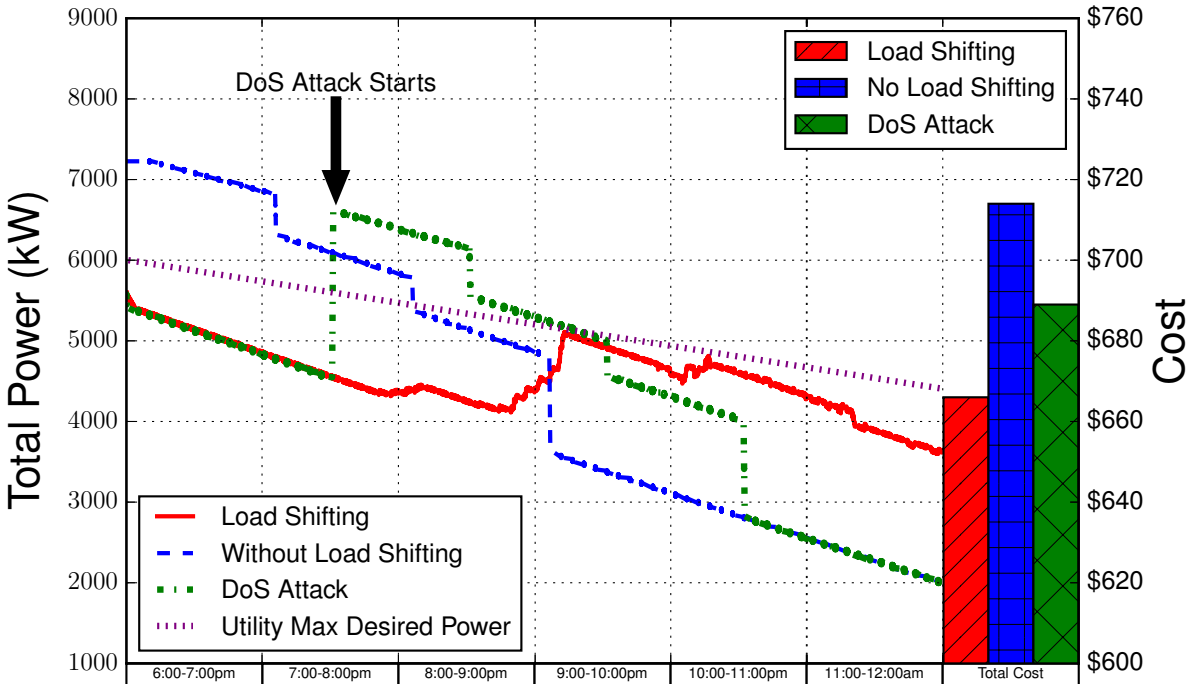
6.3 Experimental Results

Figure 11 shows the three cases of the load shifting algorithm: the distribution network total power consumption during the experiment window, with and without the load scheduling algorithm, and the load shifting algorithm under the DoS attack.

In this scenario, the utility has a different objective than the consumers. From the utilities point of view, the objective is to flatten the load curve by reducing the peak load. This can be accomplished by providing an artificial market to the consumer. The consumer on the other hand, desires to minimize the cost. In this market, price is forecasted to plan the load shifting. The price listed is the consumer price paid for power. The utilities desired total power is shown in Figure 11.

In each case, the total amount of power in the time window 6:00 pm –12:00 am is the same. The cost of power is calculated using the amount of power used in the 5-minute time slot windows. The hourly quantities are summarized in Figure 11. With the load scheduling algorithm, the total consumer cost of power in the distribution network is $666.01, while without the algorithm the total cost is $713.66. When the load shifting algorithm experiences a DoS attack, the total cost is $688.57. Even when the communication network is under attack, the overall cost has been lowered, due to partial load shifting.

Only when the load scheduling algorithm is used, does the utility see its objective met. When the DoS attack occurs, the total power from 7:30 – 9:00 pm exceeds the utility's goal. This prompts motivation to research mitigation techniques using SDN.



	6:00-7:00pm	7:00-8:00pm	8:00-9:00pm	9:00-10:00pm	10:00-11:00pm	11:00-12:00am	Total Cost
Avg Price (cents/kWh)	3.15	2.9	2.75	2.3	1.9	1.4	2.40(Avg)
Load Shifting (USD)	158.62	130.57	115.27	112.65	91.41	57.51	666.01
No Load Shifting (USD)	217.15	177.34	143.98	86.85	55.95	32.39	713.66
DoS Attack (USD)	158.62	161.85	153.56	114.26	67.89	32.39	688.57

Figure 11: Normal Power Consumption, Power Consumption with Load Shifting Algorithm, Power Consumption with DoS attack, Desired Utility Price. With the load shifting algorithm the power consumption is below the utilities desired limit.

7. CONCLUSION AND FUTURE WORK

We present DSSnet, a testing platform that combines an electrical power system simulator and an SDN-based network emulator. DDSnet can be used to model and simulate power flows, communication networks, and smart grid control application, and to evaluate the effect of network applications on the smart grid. Our future work includes exploring means to extract emulation lookahead to improve the performance of this hybrid system, as well as developing the distributed version of the testbed for large-scale experiments. We also plan to investigate several novel SDN applications for microgrid security and resilience, such as network-wide configuration verification, and context-aware intrusion detection.

8. ACKNOWLEDGMENTS

This paper is partly sponsored by the Maryland Procurement Office under Contract No. H98230-14-C-0141, and the Air Force Office of Scientific Research (AFOSR) under grant FA9550-15-1-0190. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Maryland Procurement Office and AFOSR.

9. REFERENCES

[1] Comed. https://hourlypricing.comed.com/live-prices/?date=20151106. [Last accessed December 2015].

[2] Electric power monthly. https://www.eia.gov/electricity/monthly/epm_table_grapher.cfm?t=epmt_5_3. [Last accessed January 2016].

[3] IIT campus microgrid project. http://www.iitmicrogrid.net/microgrid.aspx. [Last accessed December 2015.

[4] iperf3. http://software.es.net/iperf. [Last accessed December 2014].

[5] iputils. http://www.skbuff.net/iputils/. [Last accessed November 2015].

[6] Opendss program, sourceforge.net. http://sourceforge.net/projects/electricdss. [Last accessed January 2016].

[7] Zeromq. http://zeromq.org/. [Last accessed January 2016].

[8] A. Agnetis, G. De Pascale, P. Detti, and A. Vicino. Load scheduling for household energy consumption optimization. *IEEE Transactions on Smart Grid*, 4(4):2364–2373, 2013.

[9] S. Ciraci, J. Daily, K. Agarwal, J. Fuller, L. Marinovici, and A. Fisher. Synchronization algorithms for co-simulation of power grid and communication networks. In *Modelling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2014 IEEE 22nd International Symposium on*, pages 355–364, Sept 2014.

[10] S. Ciraci, J. Daily, J. Fuller, A. Fisher, L. Marinovici, and K. Agarwal. Fncs: A framework for power system and communication networks co-simulation. In *Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative*, DEVS '14, pages

36:1–36:8, San Diego, CA, USA, 2014. Society for Computer Simulation International.

[11] X. Dong, H. Lin, R. Tan, R. K. Iyer, and Z. Kalbarczyk. Software-defined networking for smart grid resilience: Opportunities and challenges. In *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security*, CPSS '15, pages 61–68, New York, NY, USA, 2015. ACM.

[12] C. Dufour and J. Belanger. On the use of real-time simulation technology in smart grid research and development. *Industry Applications, IEEE Transactions on*, 50(6):3963–3970, Nov 2014.

[13] R. C. Dugan. Reference guide, the open distribution system simulator, 2013.

[14] T. Godfrey, S. Mullen, R. Dugan, C. Rodine, D. Griffith, and N. Golmie. Modeling smart grid applications with co-simulation. In *Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on*, pages 291–296, Oct 2010.

[15] A. Goodney, S. Kumar, A. Ravi, and Y. Cho. Efficient pmu networking with software defined networks. In *Smart Grid Communications (SmartGridComm), 2013 IEEE International Conference on*, pages 378–383, Oct 2013.

[16] K. Hopkinson, X. Wang, R. Giovanini, J. Thorp, K. Birman, and D. Coury. Epochs: a platform for agent-based electric power and communication simulation built from commercial off-the-shelf components. *Power Systems, IEEE Transactions on*, 21(2):548–558, May 2006.

[17] Y.-J. Kim, K. He, M. Thottan, and J. Deshpande. Virtualized and self-configurable utility communications enabled by software-defined networks. In *Smart Grid Communications (SmartGridComm), 2014 IEEE International Conference on*, pages 416–421, Nov 2014.

[18] J. Lamps, D. M. Nicol, and M. Caesar. Timekeeper: A lightweight virtual time system for linux. In *Proceedings of the 2Nd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, SIGSIM PADS '14, pages 179–186, New York, NY, USA, 2014. ACM.

[19] B. Lantz, B. Heller, and N. McKeown. A Network in a Laptop: Rapid Prototyping for Software-defined Networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, pages 19:1–19:6, New York, NY, USA, 2010. ACM.

[20] H. Lin, S. Veda, S. Shukla, L. Mili, and J. Thorp. Geco: Global event-driven co-simulation framework for interconnected power system and communication network. *Smart Grid, IEEE Transactions on*, 3(3):1444–1456, Sept 2012.

[21] K. Mets, J. Ojea, and C. Develder. Combining power and communication network simulation for cost-effective smart grid analysis. *Communications Surveys Tutorials, IEEE*, 16(3):1771–1796, Third 2014.

[22] E. Molina, E. Jacob, J. Matias, N. Moreira, and A. Astarloa. Using software defined networking to manage and control iec 61850-based systems. *Comput. Electr. Eng.*, 43(C):142–154, Apr. 2015.

[23] D. Montenegro, M. Hernandez, and G. Ramos. Real time opendss framework for distribution systems simulation and analysis. In *Transmission and Distribution: Latin America Conference and Exposition (T D-LA), 2012 Sixth IEEE/PES*, pages 1–5, Sept 2012.

[24] H.-C. Sun and Y.-C. Huang. Optimization of Power Scheduling for Energy Management in Smart Homes. *Procedia Engineering*, 38:1822–1827, 2012.

[25] A. Sydney, D. S. Ochs, C. Scoglio, D. Gruenbacher, and R. Miller. Using geni for experimental evaluation of software defined networking in smart grids. *Computer Networks*, 63:5–16, 2014.

[26] J. Yan and D. Jin. A virtual time system for linux-container-based emulation of software-defined networks. In *Proceedings of the 3rd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, SIGSIM PADS '15, pages 235–246, New York, NY, USA, 2015. ACM.

[27] J. Yan and D. Jin. Vt-mininet: Virtual-time-enabled mininet for scalable and accurate software-define network emulation. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 27:1–27:7, New York, NY, USA, 2015. ACM.

[28] Y. Zheng, D. Jin, and D. M. Nicol. Impacts of application lookahead on distributed network emulation. In *Proc. of the 2013 Winter Simulation Conference (WSC)*, pages 2996–3007, 2013.