

Designing a Software Platform for Evaluating Cyber-Attacks on the Electric Power
Grid

by

Roozbeh Khodadadeh

A Thesis Presented in Partial Fulfillment
of the Requirement for the Degree
Master of Science

Approved April 2019 by the
Graduate Supervisory Committee:

Guoliang Xue, Co-Chair
Lalitha Sankar, Co-Chair
Oliver Kosut

ARIZONA STATE UNIVERSITY

May 2019

ABSTRACT

Energy management system (EMS) is at the heart of the operation and control of a modern electrical grid. Because of economic, safety, and security reasons, access to industrial grade EMS and real-world power system data is extremely limited. Therefore, the ability to simulate an EMS is invaluable in researching the EMS in normal and anomalous operating conditions. I first lay the groundwork for a basic EMS loop simulation in modern power grids and review a class of cybersecurity threats called false data injection (FDI) attacks. Then I propose a software architecture as the basis of software simulation of the EMS loop and explain an actual software platform built using the proposed architecture. I also explain in detail the power analysis libraries used for building the platform with examples and illustrations from the implemented application. Finally, I will use the platform to simulate FDI attacks on two synthetic power system test cases and analyze and visualize the consequences using the capabilities built into the platform.

TABLE OF CONTENTS

| | Page |
|--|------|
| LIST OF TABLES | iv |
| LIST OF FIGURES | v |
| CHAPTER | |
| 1 INTRODUCTION | 1 |
| 1.1 Historical Background | 1 |
| 1.2 Smart Grid vs the Traditional Grid | 2 |
| 1.2.1 Generation | 2 |
| 1.2.2 Supervisory Control and Data Acquisition (SCADA) | 3 |
| 1.2.3 Energy Management System (EMS) | 4 |
| 1.2.4 Security | 4 |
| 1.3 Network and Attack Simulation | 7 |
| 1.3.1 Simulation Taxonomies | 8 |
| 1.3.2 Simulation Goals | 9 |
| 2 PLATFORM OVERVIEW | 13 |
| 2.1 Platform Architecture | 13 |
| 2.2 Power System Analysis Modules | 17 |
| 2.2.1 OpenPA Core | 17 |
| 2.2.2 Power Flow | 22 |
| 2.2.3 State Estimation | 25 |
| 2.2.4 Contingency Analysis | 30 |
| 2.2.5 Security Constrained Economic Dispatch (SCED) | 35 |
| 2.2.6 Anomalous Data Detector | 40 |
| 2.3 Web Application | 41 |
| 2.4 OpenPA Parser | 43 |

| CHAPTER | Page |
|---|------|
| 2.5 Single Page Application | 46 |
| 3 CASE STUDIES | 49 |
| 3.1 General Attack Mechanism | 49 |
| 3.2 Attack Against Polish System | 51 |
| 3.3 Attack Against Texas Synthetic System | 54 |
| 3.4 Conclusion and Future Work | 56 |
| REFERENCES | 58 |

LIST OF TABLES

| Table | Page |
|---------------------------------------|------|
| 2.1 OpenPA Parser Mini-Language | 44 |

LIST OF FIGURES

| Figure | Page |
|---|------|
| 1.1 NIST Smart Grid Network Conceptual Diagram | 6 |
| 2.1 General EMS Loop | 13 |
| 2.2 Cybersecurity Simulation Platform Architecture | 15 |
| 2.3 UML Diagram For Class Load | 19 |
| 2.4 Piece-wise Linear Cost Modeling with OpenPA Core | 21 |
| 2.5 Power Flow Algorithm Flow Diagram | 22 |
| 2.6 Power Flow Options Form | 24 |
| 2.7 Power Flow Results | 24 |
| 2.8 Calculated Branch Flows In The Polish System | 25 |
| 2.9 Change In Slack Distribution During Power Flow Iterations | 26 |
| 2.10 State Estimation Components | 29 |
| 2.11 Bad Measurements In ACTIVSg Power System | 31 |
| 2.12 Bad Data Detection Results in ACTIVSg System | 31 |
| 2.13 Contingency Analysis Options | 34 |
| 2.14 Contingency Analysis Results | 35 |
| 2.15 Matrix Diagram For System Contingencies | 36 |
| 2.16 SCED Components Diagram | 37 |
| 2.17 SCED Options Form 1 | 38 |
| 2.18 SCED Options Form 2 | 38 |
| 2.19 SCED Results Table | 39 |
| 2.20 Dispatch differences After SCED | 39 |
| 2.21 Web Application Components | 41 |
| 2.22 OpenPA Parser Abstract Syntax Tree | 45 |
| 2.23 OpenPA Parser Inheritance Hierarchy | 45 |

| Figure | Page |
|--|------|
| 2.24 SPA Components | 48 |
| 3.1 Simplified EMS Loop Diagram | 50 |
| 3.2 Attack Consequences Diagram For The Polish System | 53 |
| 3.3 Worst Overload In The Physical World In The Polish System Attack .. | 54 |
| 3.4 Texas System Attack Consequences | 55 |
| 3.5 Geographical Location Of The Lines Affected Most By The Texas Sys- tem Attack | 56 |

Chapter 1

INTRODUCTION

In this chapter, I will first provide a historical background for the smart grid evolution and discuss the ways it is different from the traditional power grid with a focus on the cybersecurity aspects of the smart grid. I then discuss smart grid simulation and justify making certain design decisions which will define the framework of our software simulation system.

1.1 Historical Background

From its beginning in the late nineteenth century, the electrical power grid has become increasingly interconnected in response to consumer demand. For example, in 2018 alone, the US power grid distributed more than 4,000,000 Megawatt-hours of electrical power to hundreds of millions of consumers [1]. Along with the evolution of the grid itself, the question of how to control and monitor it has also become increasingly important. During the 1970s, a revolution occurred in the industrial control systems which allowed these systems to effectively and quickly communicate with each other. Electric grid operators have utilized these advances in information technology to offer more accurate sensing, faster control response, and better and more efficient grid planning and operation. In the past two decades, the combination of advanced communication and information processing infrastructure and the electric grid has often been called the *smart grid*. The European technology platform defines a smart grid as “an electricity network that can intelligently integrate the actions of all users connected to it, generators, consumers and those that do both, in order to efficiently deliver sustainable, economic and secure electricity supply [2].” In general,

all notions of an advanced power grid for the 21st century hinge on adding and integrating many varieties of digital computing and communication technologies and services with the power-delivery infrastructure. Another important property of smart grids is the bidirectional flows of energy and two-way communication and control capabilities which will enable an array of new functionalities and applications [3].

Next section briefly goes over how smart grids have added these new functionalities and capabilities to the traditional grid.

1.2 Smart Grid vs the Traditional Grid

1.2.1 *Generation*

The electricity market has seen a regulatory and societal demand in the past decade to use more renewable energy. However, in the United States, the sources of renewable energy are usually far from load centers [2]. This fact creates a host of issues for the transmission system, such as new contingencies, stability issues, transients, and power quality issues in grids that have a high share ($> 20\%$) of renewable energy sources [2]. Also, coordination between renewable sources and fast ramping rate units is needed to ensure uninterrupted power supply for consumers. With the rise in the use of electric vehicles and other novel loads, even more unpredictability is introduced into the grid operation. The decision making infrastructure of the traditional power grid might not be suitable to respond to these new challenges. Smart grids with high processing powers are needed to coordinate the consumer demand and electricity production in the modern grid.

1.2.2 *Supervisory Control and Data Acquisition (SCADA)*

SCADA consists of all the sensors installed in the field and the communication network that transmits the data from those sensors to the control center. At one end of the system lie the remote terminal units (RTU). At the other end, there are human machine interfaces (HMI) that display the data for the human operator [2].

Smart grid integration has enabled vast data gathering and processing capabilities. System operators can now gather data such as geographical information and store it for later reference. However, serious security vulnerabilities accompany these advanced functions. When SCADA systems were first equipped with advanced computer technology, there was little thought given to security. The industrial communication networks were often isolated and devices manufactured by different vendors, worked differently. The assumption was that such obscurity would ensure the security of SCADA systems. Eventually, it became evident that SCADA systems are not immune to cyber attacks. A survey of 100,000 attacks against industrial systems is presented in [4]. Many of these attacks exploit the vulnerabilities within communication networks with severe consequences including code execution, privilege escalation, denial of service, and data extraction.

The advent of the Internet of Things (IoT) has severely exacerbated the industrial network exploitation problem since the traditional factors that contributed to a secure industrial network are weakened or non-existent due to an interconnected and vast communication network. In short, the smart grid has added many new data processing capabilities to the traditional SCADA systems, but at the same time has made those systems more vulnerable to various types of cyber attacks. It is not correct to assume that SCADA is impenetrable simply because it is usually operated on dedicated or private industrial networks.

1.2.3 Energy Management System (EMS)

Smart grid network operations rely largely on EMS. EMS is a combination of hardware and software often used in control rooms and dispatching centers that adds full computer automation to all operational and analytical functionalities of a smart grid. The term “EMS” is often used in the context of a high voltage transmission system. For lower voltage distribution network, “distribution management system (DMS)” is often used. An EMS has the following functions:

- real-time grid monitoring and control: displays signals from the SCADA system for the operators and may allow them to open and close circuit breakers remotely.
- frequency control: ensures that the network operates at the designated frequency.
- system studies: state estimation, contingency analysis, security constrained economic dispatch, and security constrained unit commitment among others to facilitate efficient operation of the grid.

More discussions about the EMS role requires us to understand the concept of security in smart grids. Next section will analyze security in smart grids and explain why EMS has such a central role in smart grid operations.

1.2.4 Security

In contrast to the traditional power grid, A modern electric grid has large physical and cyber components and thus is categorized as a cyber-physical system. The term “security” can have somewhat different meanings when used in the context of power grids. On one level, the operators pay serious attention to the electrical security of

the system: they plan for scenarios in which one or more network devices fails and cannot serve its purpose and they try to plan ahead for such circumstances. This type of study is called contingency analysis (CA). The system operator plans for as many scenarios as is technically feasible. The typical case is to assume only one device has failed at a time. The operators then devise contingency plans to tackle the equipment outage problem. With such plans in place, the system is said to be $N - 1$ secure.

Contingency analysis requires the operators to have accurate the real time data from the network. Due to the presence of noise in communication channels, there is a possibility that the measurements across the system are not consistent. It is essential to run another type of analysis called state estimation (SE) to obtain the most probable measurements in the presence of noise.

Moreover, if the operators determine that the system is not $N - 1$ secure, or it is not operating in a cost-effective manner, they may conduct another study called security constrained economic dispatch (SCED) to make sure even in scenarios when some network device is out of service, the rest of the network can deliver the demand and still operate securely and economically.

$N - 1$ security can become very complex. But the notion of security can also imply actual physical safety of the electrical grid equipment since many of these equipment contain sensitive measurement technologies and some may operate at high voltage levels. To ensure public safety and preclude tampering, network equipment must also be protected by physical enclosures and barriers.

On another level, security can be considered as the cyber security of a smart grid's communication network. I discussed SCADA security in subsection 1.2.2. But the cyber component of a smart grid can be even more complex. Figure 1.1 shows a conceptual diagram created by the national institute of standards and technology (NIST). It shows the many connected networks that comprise a smart grid together[3]. Any

security breach that threatens confidentiality, integrity, or availability of any of these networks has the potential of compromising the whole smart grid. Moreover, these networks are spread over vast geographical areas and may be administered by independent entities. These properties have made designing a comprehensive protection scheme for the smart grid a major challenge.

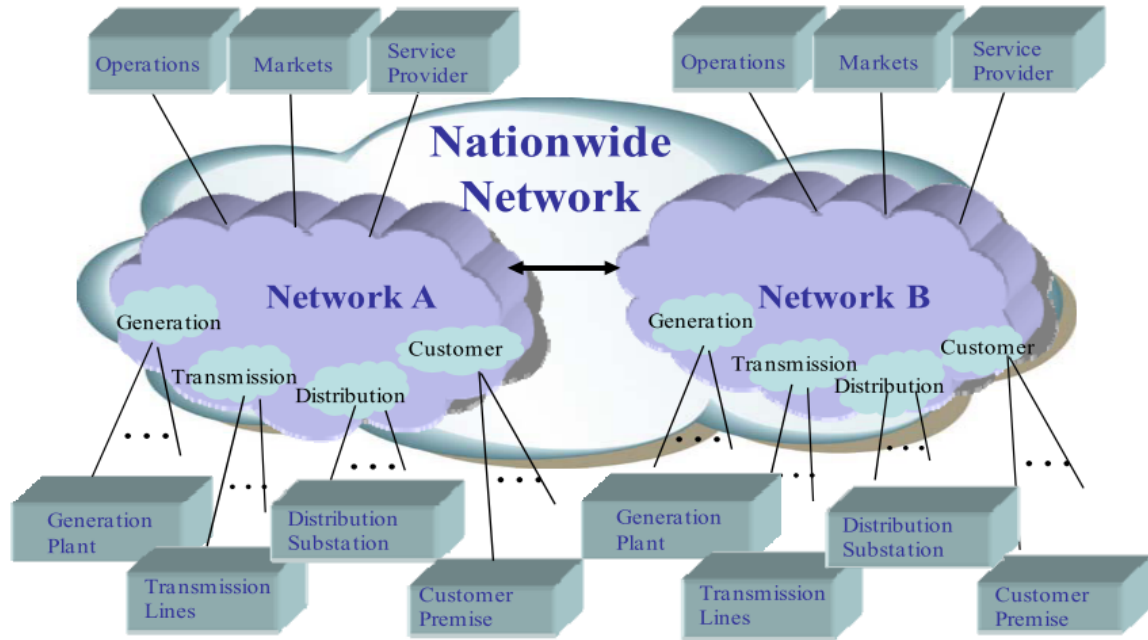


Figure 1.1: Conceptual diagram showing various network interconnections in the context of smart grids.

Still, attacking a smart grid can require a high level of sophistication. To successfully conduct a cyber-physical attack against the smart grid, the attacker needs to:

1. breach network infrastructure. Any of the interconnected networks depicted in figure 1.1 can be a target for an attacker to find a way into the smart grid cyber system.
2. The attacker ideally wants to compromise the main control system of the smart

grid. The attacker may achieve this by spoofing commands to the control system. One way to carry out such attack would be by simply sending commands that opens or closes circuit breakers. This type of attack is often detectable because it causes outages or equipment damage. Alternatively, the attacker may use her access to the main cyber system to inject false measurements into the smart grid decision-making loop in to deceive the operators into making bad decisions resulting in adverse consequences for the physical network. The latter is often called false data injection attack (FDIA). In this case, the attacker usually does not want to be detected.

An attacker may carry out an FDI attack by infiltrating either the sensor network (SCADA and RTUs), the communication network, or the EMS and control center itself. Because of large number of options the attacker has to carry out an FDIA, it is worthwhile to assume a successful FDIA and study only the consequences.

Because of the size, physical and operational complexity, and expensive and sensitive equipment, it is usually infeasible to conduct the security studies on a running industrial smart grid. Researchers often use simulation to safely analyze attacks and their consequences without endangering the sensitive operations of the grid.

1.3 Network and Attack Simulation

Shannon [5] defines simulation as “the process of designing a model of a real system and conducting experiments with this model for the purpose either of understanding the behavior of the system or of evaluating various strategies (within the limits imposed by a criterion or set of criteria) for the operation of the system.” This definition is very broad. In this section, I first define a set of constraints which simplify the power system as a cyber-physical model and enable us to simulate it with computer software. The simplifications resulting from applying these constraints still allow

efficient modeling and analysis of false data injection attacks with high precision.

1.3.1 *Simulation Taxonomies*

Depending on the simulated aspect of the system, many simulation taxonomies are devisable. Here, I use some of the taxonomies Shlooeagl [6] has defined. Based on [6], one possible simulation taxonomy is:

- A single model representing the whole system running within a dedicated run-time environment
- Parallel models running across multiple solvers; there is still one representation
- Hybrid simulation which integrates more than one representation into a single run-time environment
- Co-simulation in which different models with different representations are executed inside their own dedicated run-time environments. There needs to be some form of synchronization between co-simulation components.

Another way [6] to categorize smart-grid simulations is to look at how the simulator approaches simulation timing:

- Real-time: the simulator guarantees a constant delay between the events and the simulation of those events. Events are usually time-stamped to ensure correct order of execution
- Non Real-time: there is no guarantee of constant delay

The notion of being on-line or off-line may also be used to classify simulators [6]:

- Online : the simulator interacts with outside physical processes between simulation steps

- Offline: measurements and outside processes are also simulated and there is no interaction between the simulator and the actual physical process it is simulating

Time resolution provides an additional method to classify simulators [6]:

- Steady state range: the system is in equilibrium.
- Electro-mechanic range: (sub-seconds): generators are dynamically interacting, and frequency transients, stability and short circuit behavior can be analyzed.
- Electro-magnetic Range (milliseconds and below): electromagnetic waves are the dominating mechanism; fast transients, dynamic voltage problems and stability are examples.

These taxonomies provide a clear way to think about the simulation model. By carefully selecting taxonomy levels from the above definitions, simulation constraints and ultimately the simulation model is precisely determined. Choosing a different simulation configuration may have profound implications on the simulation complexity and feasibility, therefore, a more in-depth discussion of the simulation goals is needed.

1.3.2 Simulation Goals

The ultimate purpose of the simulator is to analyze FDI attacks carried out against smart grids. While false data injection attacks can be used to compromise the system at any level, even the electro-magnetic range [7], here the focus is on a class of FDI attacks described in [8]. It is a sophisticated attack that calculates a set of false measurements by formulating a bi-level optimization problem. The optimization problem maximizes the attacker gain assuming the operator makes the best possible decisions while trying to satisfy $N - 1$ security. In this context, operator action is defined as setting the output power from the system generators. As described in

subsection 1.2.4. the study that determines the best dispatch point is the SCED. The bi-level optimization problem gives a set of system measurements that if successfully spoofed, would lead to SCED calculating a set of dispatch points which in turn would compromise system's $N - 1$ security.

The attack in [8] is carried out in the steady-state timing level. The communication system is assumed to deliver all the measurements to the control center instantaneously and simultaneously and therefore is not simulated. Consequently, the simulator can be designed to only simulate events in the steady-state timing level.

Another implication of the steady-state assumption is that the events in the simulated system occur sequentially. In other words, there are no concurrent events in the system. If two events happen at the same time, one should have priority before feeding them to the simulator. This is effectively a discrete event simulation loop (DES) in which the simulator jumps from one event to another without considering any other event between the two. Some level of concurrency can be added to DES loop by assuming that the simulator advances in fixed time intervals. This is called discrete time simulation (DTS). With DTS, all events occurring in a certain time interval are assumed to have happened exactly at the same moment.

To give an example from the power system, if the operators decide to open two circuit breakers in two different locations in the network at the same time, A discrete simulation model completely ignores the transient effects of such openings. Instead, the system is simply modeled as if one circuit breaker opens a long time before the other (DES), or as if both are open at the exact same time (DTS) without having any transient effects. As far as the attack in [8] is considered, a DES model suffices. Simulation time moves only when a new event - whether in cyber of physical world - occurs. Since switching operations are not simulated, there is no need for real-time simulation.

The role of simulation engine users also plays an important role in determining the simulation category. For example, a training platform may allow the users to directly interact with the power system and simulate their actions. FDI attacks are usually carefully designed attack scenarios that the attacker executes amidst the normal EMS event loop. As a result, an off-line simulation model is preferred to the on-line model. The events are predetermined and are fed to the system sequentially. To understand this distinction, the off-line simulation model can be compared with flight simulators. In a flight simulator - much like a power system training simulator - the actions of the user will have real-time consequences on the simulated world. This is not the case with FDI attack simulation. In other words, there is no human user in the simulation loop.

To summarize, given the criteria for FDI attack studies, the simulation is limited to the **discrete event**, **non-real-time**, and **offline** simulation of power system events.

The first implication of picking discrete event simulation (DES) loop as the simulation basis, is that only steady-state power analysis libraries are needed. For example, a package such as MatPower [9], which is a widely used power system analysis software, could be used. As the system jumps from event to event, the analysis library is used to update system state. The performance of the analysis library is not the top priority because there is no real-time events.

Ideally, an open source software should be used to make every part of the code accessible to programmers or researchers who wish to study or extend it. It is also desirable to use a software that has at least some real world and industrial usage to obtain a better understanding of the power system industry's perspective and mindset.

MatPower can only be executed in MATLAB environment which itself not free and open source software. It is also not widely used in the industry. Instead, to

build the platform, an industry partner (IncSys), which is active in training power system operators, was chosen. The collaboration with IncSys had a twofold goal: improve the performance and quality of their power analysis libraries, and to build a full-fledged simulation platform on top of those libraries. IncSys uses Java [10] programming language to develop their power analysis library and modeling. Java is a general-purpose, fast, object-oriented programming language with a huge set of 3rd party libraries, and works very well with web technologies. IncSys' power analysis library is called OpenPA [11].

In the next chapter, the software system platform based on the criteria laid out in this section is described. The usage of this software system to conduct various studies and download comprehensive reports is also illustrated.

In chapter 3, I will present two case studies to show how sophisticated FDI attacks can be carried out against a power system and how the platform helps the user quickly and easily understand the attack process and its consequences.

PLATFORM OVERVIEW

This chapter describes in detail the architecture of the designed software platform. The designed platform was then implemented to test EMS operations and various cyber attacks. The starting point the discussion is the conceptual event loop, which is then translated into a software architecture. Examples of how each building block works by directly running experiments using the deployed platform are also given.

2.1 Platform Architecture

Figure 2.1 shows the concept and data flow in the general EMS loop. This loop is the conceptual basis for the architecture design, although it is important to note that an industrial-grade EMS may have many more components than shown in figure 2.1. The final software platform is capable of modeling the loop in figure 2.1 and also allows building new and higher-level functionalities on top of it.

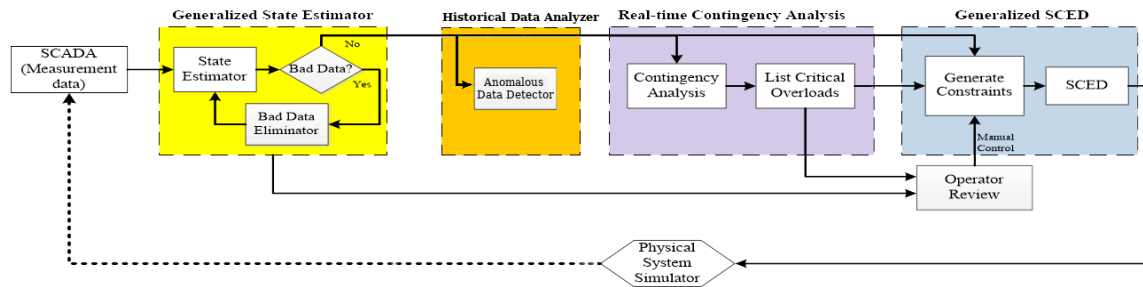


Figure 2.1: General EMS loop: solid arrows show the data flow in the cyber system. Dashed arrow shows the concept of SCADA acquiring measurements from the physical system.

The first step is to transform the simulation goals laid out in chapter 1 into a list of requirements that can be met by a software system:

- The platform should be able to simulate the loop in figure 2.1 using a discrete event simulation (DES) model. Time advance will be initially simulated by the user clicking the next step.
- The platform should be able to run the power system studies and other functions in a reasonably short time (ideally as fast as an industrial EMS system).
- The platform should be able to efficiently process power system case data, load the case and make all the data in the case available to the client code in an appropriate format (graph, matrix, semantic objects, etc)
- The platform should be able to run optimization software capable of solving linear programming models for the SCED study.
- The platform should be able to present the system and the results of different simulation stages in comprehensive reports or visualizations to the user.
- The platform should have a modular design to facilitate later addition of new functionalities.

For the rest of this chapter, unless stated otherwise, C4 modeling language is used to describe the software systems and its components. C4 (c4model.com) defines four levels for a software architecture:

1. Context: defines the boundaries of the system and how it generally interacts with the users.
2. Container: zooms into the software system and shows the main building blocks that compromise the software system.
3. Component: zooms into individual containers and shows the inner blocks of a container.

4. Code: very detailed view of each component.

Using the concepts in C4, figure 2.2 shows a container-level view of the simulator.

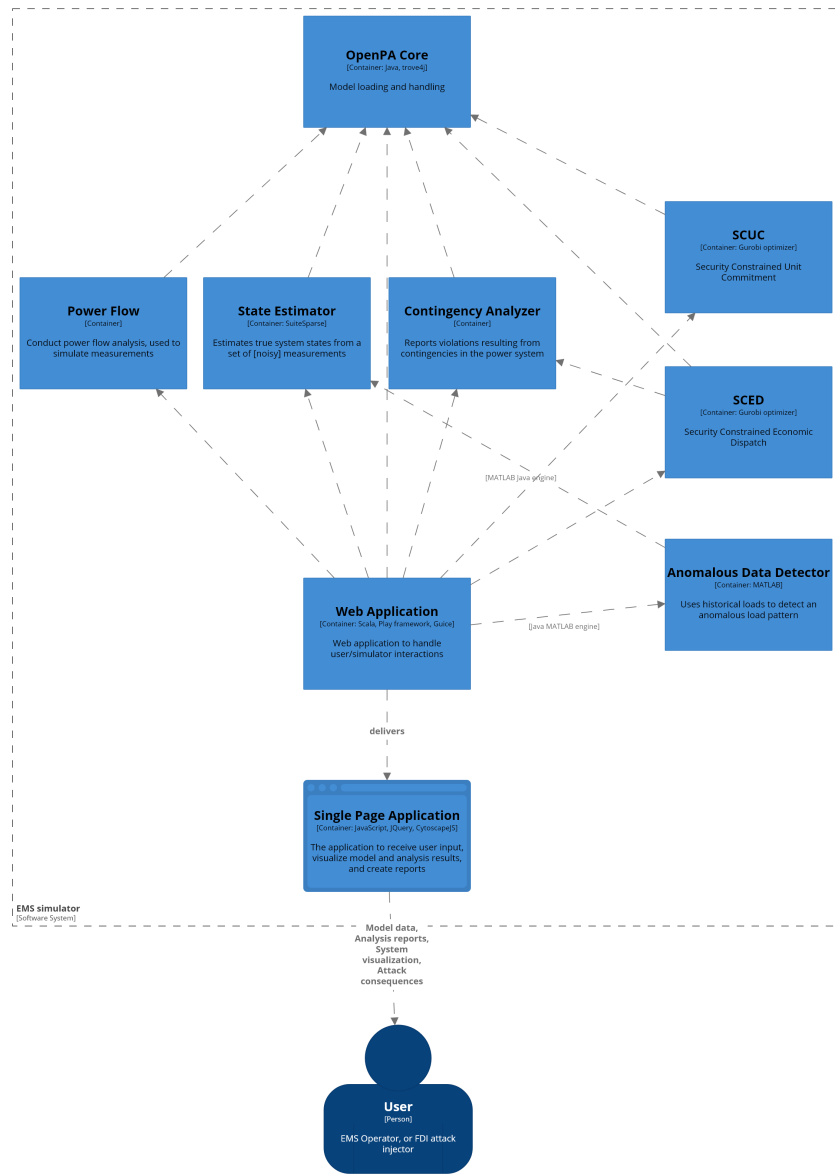


Figure 2.2: Cybersecurity simulation platform architecture using C4 modeling language. Unless labeled otherwise, the arrows denote a “uses” relationship between the containers. Each blue box denotes a “container” in C4 language.

In figure 2.2, the function of each container is printed in boldface; the technologies used to create the container are listed in brackets below the title; and a brief descrip-

tion of the container is given in the bottom of each container. To summarize, this is a client-server application ultimately delivering a single page application (SPA) to the user. The modules in figure 2.2 can be divided into two broad categories from a conceptual point of view: modules that make power system functions (e.g. power flow) available, and modules that provide user interface functionalities and transfer data (e.g. web application).

Power system modules in diagram 2.2 largely depend on the software package from IncSys called OpenPA [11]. It is important to note that the term “OpenPA” is used to describe a collection of software packages some of which can be used independently of others. At the highest level, OpenPA can be seen as a collection of three packages written in Java [10] programming language:

- OpenPA core: responsible for reading and storing the model in memory as well as providing an interface for the model to interact with the power system algorithms.
- OpenPA power library: power flow, state estimation, and contingency analysis algorithms reside in this package
- OpenPA tools: matrix operations, mathematical functions, case file format definitions, and PSSE and ohter format converters. The `Tools` package is not shown in the architecture diagram to avoid clutter and is mostly omitted from detailed discussion unless necessary.

To avoid ambiguity, I refrain from using the term “OpenPA”. Instead, I explicitly refer to the module being used in the context of the simulation platform regardless of the party who has developed that module or the module’s exact code hierarchy. Going back to figure 2.2, four modules were originally developed by Incsys: OpenPA core, power flow, state estimation, and contingency analysis. I have collaborated with

IncSyst and other team members to improve and debug these module and have also contributed the rest of the modules in figure 2.2.

2.2 Power System Analysis Modules

In this section, a review of power analysis library modules provided mostly by the industry partner (IncSyst) is presented. Understanding the structure and inter-communication of these modules is crucial to obtain an accurate idea of how the platform operates. Such knowledge also facilitates future development efforts.

2.2.1 *OpenPA Core*

As its name implies, this module sits at the core of all the platform operations. This module efficiently stores, queries, and presents the data, equipment information, and the network structure of the power system.

OpenPA core uses a sparse graph data structure called LinkNet [12]. LinkNet divides the network graph into nodes, branches, and topology. Nodes and branches are stored as indexed lists, and the topology is stored by defining links between the appropriate node and branch indices.

Podmore [12] describes in detail the implementation of LinkNet. He also proposes algorithms to count connected islands in the graph, find a certain bus, branch, or connection list of a bus, and calculate and triangulate the Jacobian matrix during the fast decoupled power flow analysis. These algorithms have been implemented in OpenPA core.

I have not compared the performance of LinkNet data structure and its associated algorithms to other available sparse graph data structures. On the other hand, I have never experienced a performance hit that could be directly related to the usage of LinkNet.

Object Oriented Modeling of the Power System

LinkNet is useful and efficient in solving network equations. But it was originally designed using FORTRAN programming language. OpenPA core additionally provides an object oriented model to represent the network. This representation abstracts over LinkNet operations and case data loading and provides a unified interface to communicate with the network. For example, to obtain a list of all the lines connected to a bus with a given ID (`busID`), one can use the following code:

```
LineList lineList = model.getBuses().getByID(busID).getLines();
```

OpenPA core enforces unique ID constraint on its model objects. Each element defined in any of the case files must have a unique string ID for the OpenPA core to correctly identify it, even if the elements are not of the same type. The ID of a transformer cannot be identical to the ID of a transmission line. It is the modeler's responsibility to ascertain that the input data files satisfy the unique ID constraint.

The core package has another package inside it named `impl`. This package is responsible for implementation of the abstractions defined in the core package. To understand the relationship between these packages, modeling of loads in the power system is analyzed. Figure 2.3 shows how loads are modeled in OpenPA.

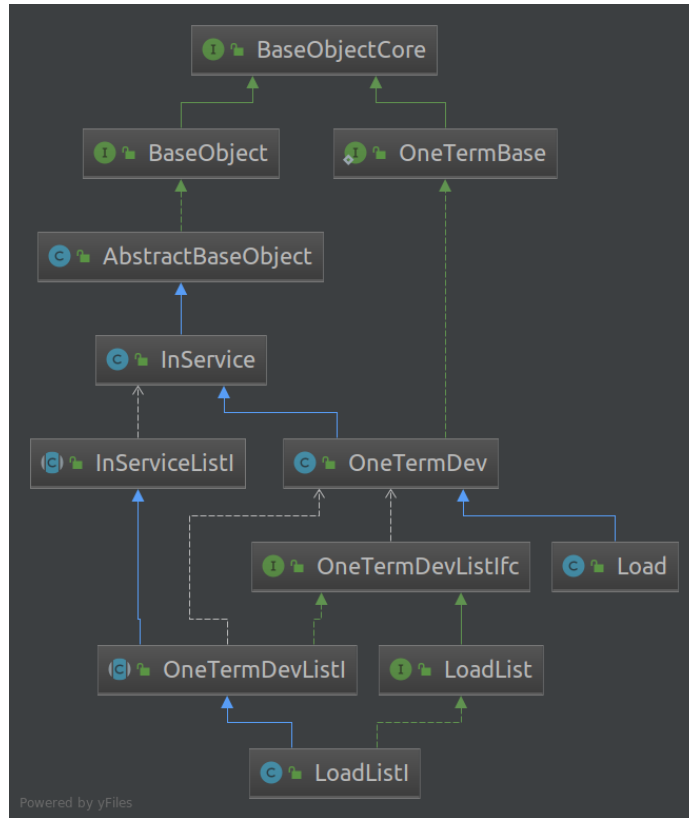


Figure 2.3: Diagram for class `Load` hierarchy in OpenPA core using unified modeling language (UML). A (C) icon denotes class, a (|C|) icon an abstract class, and a (I) icon denotes an interface. Dashed lines denote interface implementation and solid lines denote inheritance. An unlocked lock icon means these classes or interfaces have public access. Blue color is associated with classes, green color with interfaces, and white is associated with generic containers.

The object-oriented concepts such as inheritance is explained in [13]. In short, a class is a template that can hold both state and behavior, while an interface can only define behavior. Note that during model loading, OpenPA core actually only instantiates a `LoadListI` object. However, this object will have all the properties and methods of the classes in its inheritance chain up to `BaseObjectCore`. Depending on the usage, an instantiation (allocating memory) of a concrete class (colored green in figure 2.3 effectively means a union of all the classes up to the topmost class in the hierarchy chain is created.

Also the load list object is loaded lazily, that is, it is only retrieved when a method requests data from it for the first time and then remains in memory. OpenPA core provides many object properties at list level. If the user wants to obtain data for a single load, they should instantiate that load by using its index or its ID in the load list. This design improves caching performance and general efficiency of model loading and data retrieval.

The `LoadListI` object inherits from `OneTermDevListI` which in turn is a container for `OneTermDev` objects. OpenPA objects are mostly co-variant. To see what this means, the relationship between the `Load` object and its container (`LoadList` in figure 2.3) should be analyzed. A `Load` object is a sub-type of `OneTermDev`, and also a list of `Loads` is a sub-type of List of `OneTermDevs`. In other words, if class A is a subclass of class B, a container of A's is also a subclass of a container of B's.

Other power system objects also follow a hierarchy similar to figure 2.3. To augment the model with new properties, the programmer will need to add new abstract classes and associated methods in the core package and also create actual implementation of the new model objects in the `impl` package. As explained above, using inheritance helps the programmer add existing behavior in OpenPA core to her new modeling objects. Due to co-variance in OpenPA core, properties are usually implemented both at single object level and at list level. For example, figure 2.3 shows that a power system load should be modeled as a `Load` object but the programmer will also need to model `LoadList` to add list-level behavior to the modeled object. Modeling also involves figuring out what inheritance relationships need to be defined between the new model object and existing OpenPA classes. Finally, the programmer will need to define the new data field as a column from one of OpenPA CSV files and add code to the model loader to read the data correctly from the case files.

A power system is very complex and implementing new algorithms may necessitate

adding new properties to the model. For example, when my team wanted to develop a SCED module, I added piece-wise linear cost and ramp rate parameter support to OpenPA core. Figure 2.4 shows a simplified UML diagram of the final model.

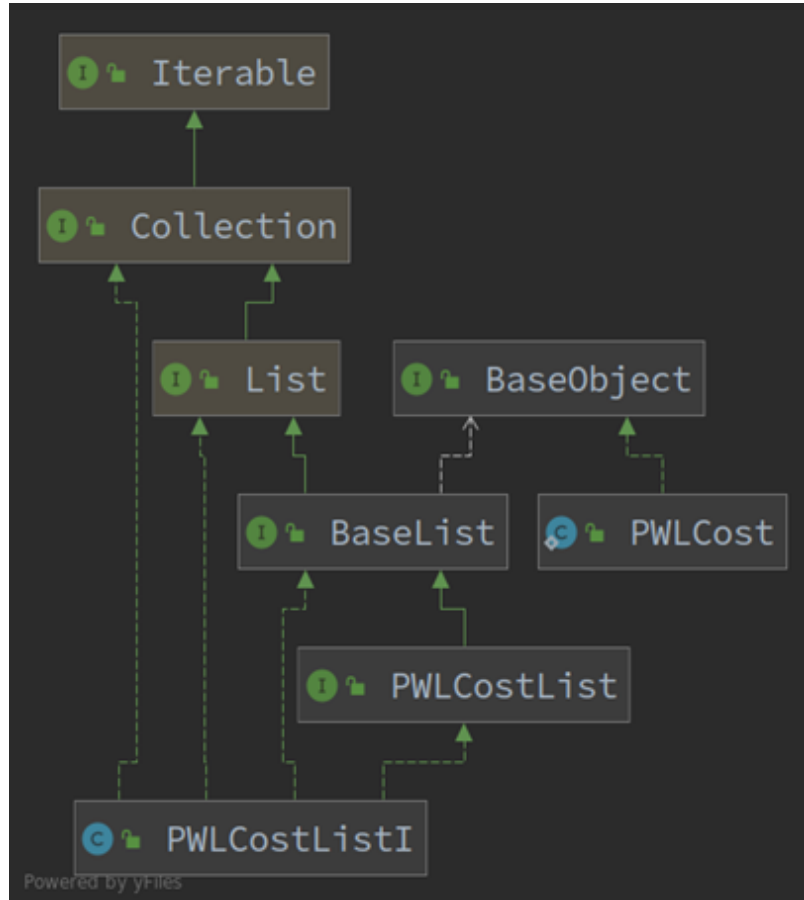


Figure 2.4: Piece-wise Linear Cost Modeling with OpenPA Core

The diagram in figure 2.4 has been simplified to emphasize the power of using inheritance in extending OpenPA core. In this diagram, I created only `PWLCost`, `PWLCostList`, and `PWLCostListI` classes. The rest of the behavior was added automatically by adding inheritance relationships with existing OpenPA core classes.

Note that the programmer can always maintain new model properties by writing code that reads the data from a separate file and synchronize that data with the OpenPA core model objects. However, adding direct support to OpenPA core for

property loading is the preferred practice since it allows OpenPA core to efficiently manage the new property. Making OpenPA core responsible for all model handling operations also makes maintaining and debugging the code easier in the future.

2.2.2 Power Flow

OpenPA implements a fast decoupled power flow algorithm described in chapter 3 of [12] and in [14]. The general flow of the algorithm is shown in figure 2.5.

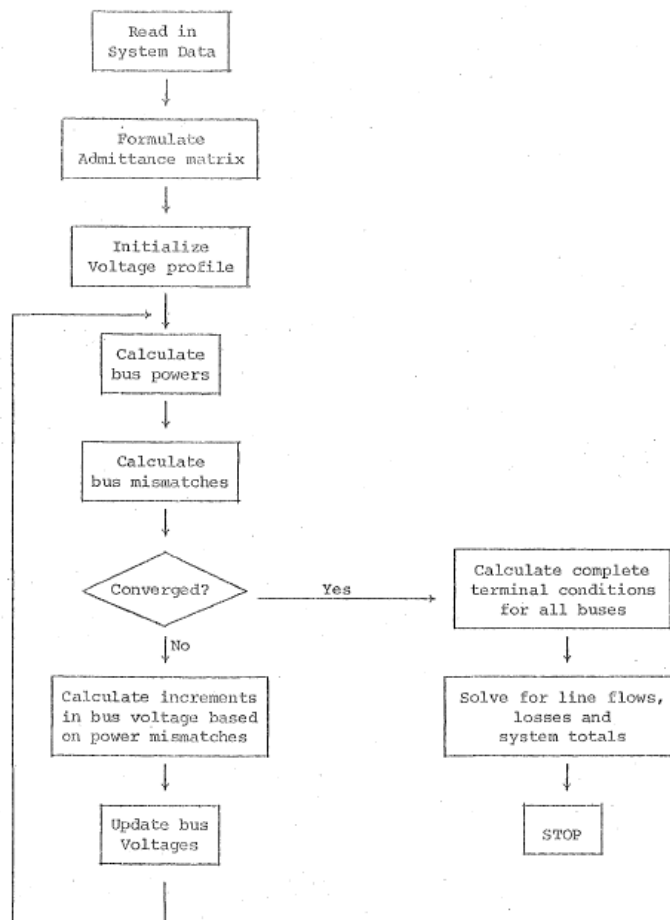


Figure 2.5: Power Flow Algorithm Flow Diagram

OpenPA power flow (OpenPA PF) adds a few adjustments and improvements to

this general algorithm. OpenPA PF allows the programmer to configure the threshold for convergence and if the mismatches are close to this threshold, it starts distributing the slack mismatch among generators where each generator picks up slack proportional to its maximum operating active power output. It also does not allow generators to exceed their capability limits during this process.

In addition, OpenPA PF detects if there are multiple energized islands in the system and runs the power flow for each island separately and reports the results. During this process, OpenPA PF automatically assigns reference buses to each island. OpenPA PF chooses reference buses based on node degree of the bus and its generation capability. OpenPA PF also monitors SVC (Static Var Compensator) devices and applies appropriate changes to the B'' matrix as the algorithm progresses.

Testing Power Flow With the Platform

For this section, a power flow study is conducted on the Polish system using the platform. This case represents the Polish 400, 220 and 110 kV networks during winter 1999-2000 peak conditions. It is part of the 7500+ bus European UCTE system. To decrease the number of buses, the tie lines to foreign networks were replaced by artificial load or generator buses (180-186). Multiple generators at a bus have been aggregated [9].

Figure 2.6 shows power flow configuration options available in OpenPA PF which the platform presents to the user via an HTML form. Figure 2.7 shows the results after the power flow algorithm converges. The user can choose to render the system graph based on various criteria. For example, figure 2.8 shows the power system graph with branch widths reflecting line flow.

If the user wishes to investigate the power flow algorithm more, they can do so by downloading full power flow report. I used this report to show how OpenPA PF

Power Flow Options

Flat Start

Lock SVCs

Convergence Tolerance

Distributed Slack Tolerance (MVA)

Maximum Number of Iterations

Slack Activation Tolerance (MVA)

Unit Active Limit Tolerance (MW)

Create Summary Report

Slack Bus List (leave empty for auto)

Figure 2.6: Power flow options form

| Island ID | Status | Gen. MW | Gen. MVA | Load MW | Load MVA | Worst V. Bus | Worst Voltage |
|-----------|-----------|----------|----------|-----------|----------|--------------|---------------|
| 0 | Converged | 20941.53 | 4768.59 | -20539.84 | -6584.81 | 1905 | 0.89 |

Figure 2.7: Power flow results in the platform user interface

implements its distributed slack algorithm. As mentioned above, instead of simply assigning all the slack to a single reference bus, OpenPA PF distributes it among generators based on a capacity profile it builds for each island. Figure 2.9 shows how some of the generators pick slack at each iteration.

In figure 2.9 each bar shows the total amount of slack picked by a generator during all iterations (6 in this case). Each colored section of each bar shows the amount of slack picked during a specific iteration. These colored sections are named “Change Slack - i” in the legend. For example, it can be seen that the generator at bus 18 picks more than 45MW of the total mismatch during the course of the power flow. Again, this is in stark contrast to some traditional power flow algorithms that simply assign all the mismatch to one generator.

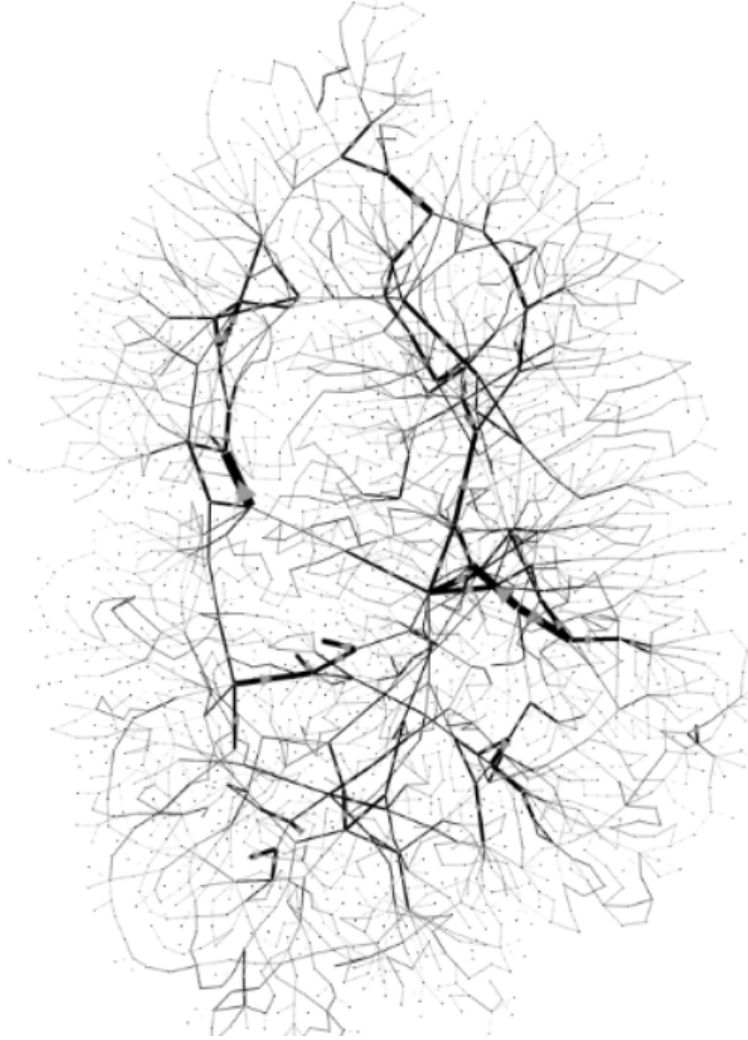


Figure 2.8: Branch flows after power flow in the Polish system. Higher line widths denote larger flow values through the branch.

2.2.3 State Estimation

OpenPA provides an state estimation module (OpenPA SE). Given a set of system measurements \mathbf{z} , set of states \mathbf{x} , state estimation problem can be formulated as (bold letters denote vectors, capitalized bold letters denote matrices):

$$\mathbf{z} = h(\mathbf{x}) + \mathbf{e} \quad (2.1)$$

$$\min F(x) = (\mathbf{z} - h(\mathbf{x}))^T \mathbf{W}(\mathbf{z} - h(\mathbf{x})) \quad (2.2)$$

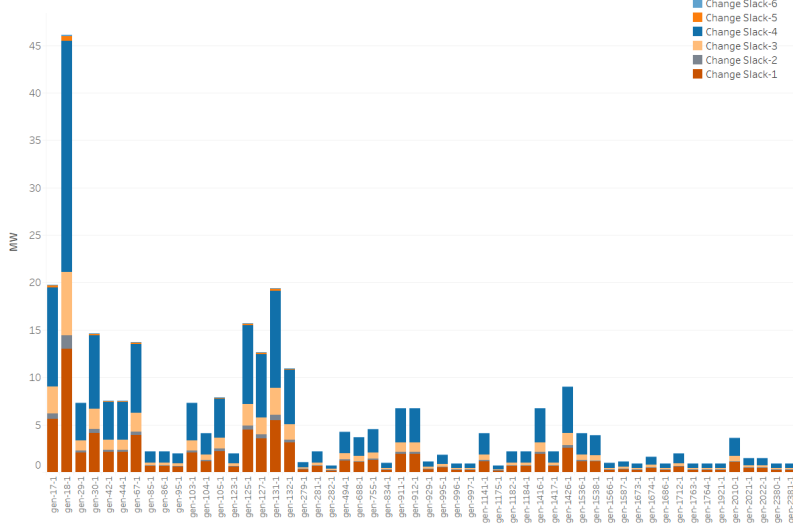


Figure 2.9: Change in slack distribution after each iteration in power flow algorithm: each bar shows the total slack picked by the generator during the course of the power flow. Each colored section shows the slack picked in that specific iteration.

The solution is:

$$\left. \frac{\partial F(\mathbf{x})}{\partial \mathbf{x}} \right|_{\mathbf{x}=\hat{\mathbf{x}}} = -2\mathbf{J}_h^T(\hat{\mathbf{x}})\mathbf{W}((\mathbf{z} - \mathbf{h}(\hat{\mathbf{x}}))) = 0 \quad (2.3)$$

In the above equation, \mathbf{J} is the Jacobian matrix, and \mathbf{W} is the weight assigned to noise level for each measurement. The objective of the above formulation is a form of weighted least square formulation where the objective is to minimize the sum of the squares of the weighted deviations of the estimated measurements, \hat{z} , from the actual measurements, z [15]. One problem with the above formulation is the existence of zero-injection buses (buses that have neither load or generation). For these buses, there is actually no need to estimate the state because it is perfectly known. If instead, a large standard deviation for noise is assigned to the measurements at these buses to force the estimate to zero, the solution matrix will become near singular. OpenPA SE uses QR decomposition [15] to address the aforementioned issue. OpenPA SE depends on SuiteSparse [16] library for sparse matrix factorization operations.

If the SE converges and a solution is found, OpenPA SE tries to assign those

measurements to the network elements. The main idea behind the state estimation is to find a power flow that best fits the measurements. Still, due to noise, there could be some discrepancies in the resulting power flow. OpenPA SE reconciles these discrepancies by first changing loads and then generator outputs if there are also generators in the buses where OpenPA SE distributes extra injections. Thus, in effect, OpenPA SE also acts as a load estimator.

OpenPA SE also contains a bad data detection which I originally developed based on the code other team members at Arizona State University had contributed. This module calculates the sum of the squares of the residuals (the weighted differences between estimated and actual measurement) after the convergence of the algorithm and compares this sum with a threshold obtained from the inverse Chi-Square distribution. Inverse chi square distribution takes a probability value and degrees of freedom for the system and returns a threshold for the acceptable error calculated above. The probability is usually set at 95%. The degrees of freedom is obtained by:

$$degrees_{freedom} = n_{meas} + n_{0injmeas} - n_{state} \quad (2.4)$$

in which n_{meas} denotes number of measurements, $n_{0injmeas}$ is the number of zero-injection measurements (buses where there is no load or generator), and n_{state} stands for the number of states (which is usually all the voltage magnitudes and voltage angles). Note that we explicitly include zero-injection buses in calculating degrees of freedom because despite having full confidence in the injection values of zero-injection buses, we still model them basically as measurements.

If a *bad* measurement is detected, state estimation is executed again after removal of the found bad measurement. OpenPA SE repeats this loop up to 50 times before stopping and reporting all the found *bad* measurements. Figure 2.10 shows a general overview of the SE package main components. In addition to these components, SE

package also performs observability analysis. However, observability analysis has not been used in the platform. Figure 2.10 shows the architecture of the main state estimation loop. Every time a solution is found, bad data detector tests the solution and sends back the results of the test to the solver.

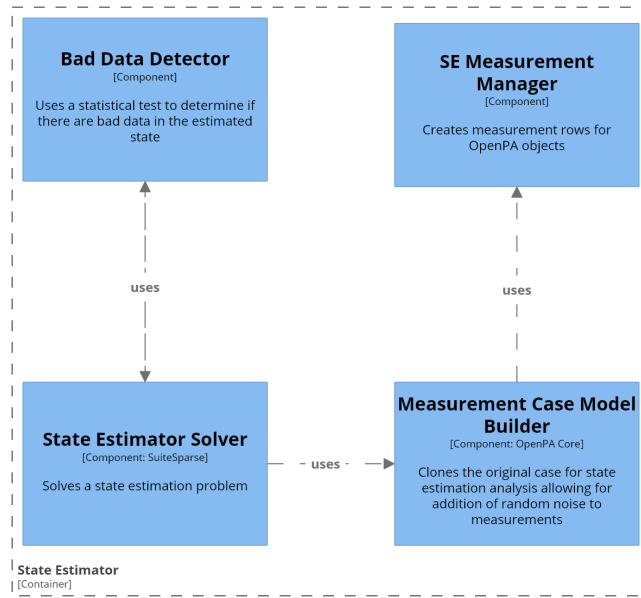


Figure 2.10: State Estimation components (based on C4 modeling language)

OpenPA SE gets its measurements from a set of pre-calculated values in the case files. The case files also contains the confidence values for each measurement. Since the noise generation algorithm simply adds Gaussian noise, confidence value is the factor that determines how much noisy a measurement can get. Consequently, since zero-injection buses have no device attached to them, there is absolute confidence that the measurements on those buses cannot be noisy. Instead of setting some arbitrary large confidence value to reflect this fact, OpenPA core has a Boolean property for such measurements named `IsTelemetered`. By setting that property to a value of false, OpenPA SE solver will use a default value of 10^{-4} for the noise standard deviation of such non-injection buses in the QR algorithm. Therefore the programmer does not need to explicitly assign confidence values to zero-injection buses like some other state estimation packages.

Testing the Bad Data Detector With The Platform

The performance of the bad data detector can be demonstrated in action by manually corrupting a few measurements in the system. The platform allows the user to download flow or injection measurements at any moment. The user can also upload data to the case by choosing “Upload Data to Model” command from the menu bar.

I used a slightly modified version of the ACTIVSg system [17] for this test. This case is a 2000-bus power system test case simulating the Texas electrical grid and is entirely synthetic. It is built from public information and a statistical analysis of real power systems.

Figure 2.11 shows a plot of the Texas synthetic system generated by the single page application in the platform with the lines with corrupted measurements highlighted in red. Figure 2.12 shows a screen shot of the SE results with the found bad data. Although only four flow measurements were changed (from-side active power and to-side active power of the two branches) the bad data detector also found that these changes would lead to a mismatch in the injection of bus 7400. The user can also investigate more by clicking an element in the table to zoom on the clicked element.

2.2.4 Contingency Analysis

OpenPA has a real time contingency analysis (OpenPA RTCA) module. RTCA gets a set of contingencies (devices to set out of service) and runs power flow for each contingency. It then reports all the violations in the network resulting from each contingency. OpenPA RTCA relies on power flow and is capable of handling contingencies that lead the network to be divided into multiple islands. OpenPA RTCA is also capable of running the power flows in parallel which means it performs better if there are more CPU cores available to it. Using RTCA with code is very



Figure 2.11: ACTIVSg (Texas synthetic) system network graph. The red branches are the branches with the bad data

| ID | Type | Estimated Measurement | Telemetry measurement | Sigma |
|----------------|-------------|-----------------------|-----------------------|-------|
| 7400 | BusInjP | 563.2378 | 566.142 | 0.01 |
| In-7002-7227-1 | BranchToP | -1082.657 | -1132 | 0.01 |
| In-7002-7227-1 | BranchFromP | 1086.7565 | 1132 | 0.01 |
| In-7366-7400-1 | BranchFromP | 1148.6355 | 1100 | 0.01 |
| In-7366-7400-1 | BranchToP | -1146.8682 | -1100 | 0.01 |

Figure 2.12: The results of running state estimation and bad data detector on the ACTIVSg (Texas synthetic) system.

straightforward. The programmer just needs to define her own RTCA class and extend it from the existing `BasicContingencyManager`. Such class will automatically be executable.

OpenPA RTCA has a `report` method which collects and processes the result of every power flow run as they conclude. Since every object in OpenPA system has a

unique ID, a hash table is a natural way of storing the contingency results. The key in this table will be the ID of the contingency and the values are the alarm reports.

Since power flows are executed in parallel, storing the results in a regular map could lead to data corruption. To avoid concurrency issues in client applications, I have used a concurrent hash map (from Java programming language) to store the results of parallel execution of RTCA. A concurrent hash map automatically synchronizes the keys and prevents simultaneous writing to the same key.

After running a power flow for a certain contingency, OpenPA RTCA detects network violations based on configurable thresholds. OpenPA RTCA can also assign MONITOR, WARNING, or VIOLATION labels depending on the severity of the actual violation. To generate the report and determine which label should be assigned to each contingency, OpenPA RTCA uses the following definitions:

1. Long term rating (LT rating): the loading under which the line can safely operate up to 24 hours.
2. Short term rating (ST rating): the loading under which the line can safely operate up to 4 hours.
3. Emergency rating: the loading under which the line can safely operate up to 15 minutes.

The above line ratings are stored in the OpenPA case files and are loaded on request by OpenPA core. OpenPA RTCA also assigns default threshold values to alarm labels. These threshold values can be configured using the RTCA builder:

1. MONITOR: set to 90% by default.
2. WARNING: set to 90% by default.

3. VIOLATION: set to 100% by default.

At the start of a contingency analysis, OpenPA RTCA first runs a base case (without any contingency) power flow. For the base case power flow, LT ratings are used to determine branch overloads along with the default threshold values. Afterwards, OpenPA RTCA iterates over the contingency list, removes each contingency and runs another power flow. For this stage, emergency ratings are used along with default thresholds. For example, assume a line has a LT rating of $90MVA$, and an emergency rating of $104MVA$. A line loading of $90MVA$ in the base case will raise a VIOLATION alarm, but it will not raise any alarm if there is a contingency. On the other hand, a line loading of $104MVA$ will raise a VIOLATION alarm both in the base case and the contingency case.

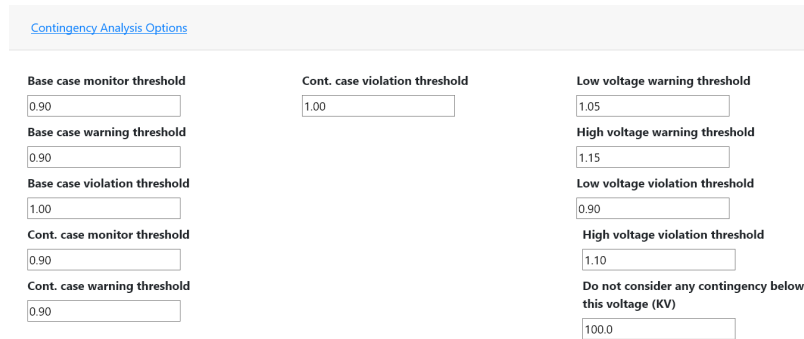
Obviously, OpenPA RTCA module depends on the power flow module described in the power flow section. If the programmer wishes to run OpenPA RTCA with different power flow settings than OpenPA PF, they can create a configuration object and pass it to the RTCA builder.

RTCA's main usage in the platform is to provide security constraints for the SCED module. To avoid problems with contingencies that lead to creation of multiple islands in the network, RTCA should remove such contingencies. I have coded and used a bridge finding algorithm that traverses the network branches, finds, and eliminates all bridge branches using Tarjan's bridge finding algorithm.

Moreover, depending on the system under study, the programmer may want to filter contingencies based on given criteria. I created a fluid interface using the builder pattern for the RTCA that allows the programmer to configure all the parameters. The filtering works by the programmer passing a predicate (e.g. *voltagelevel* > $300KV$) to the RTCA builder. This function is applied to the contingencies and to the result set to filter out any item that does not satisfy the predicate.

Testing Contingency Analysis With the Platform

In this subsection, a contingency analysis on the Polish system is conducted. Figure 2.13 shows the RTCA configuration form which the user can use to define threshold levels for alarms.



The screenshot shows a web form titled "Contingency Analysis Options" with three columns of input fields. Each field contains a numerical value.

| Field Name | Value |
|---|-------|
| Base case monitor threshold | 0.90 |
| Base case warning threshold | 0.90 |
| Base case violation threshold | 1.00 |
| Cont. case monitor threshold | 0.90 |
| Cont. case warning threshold | 0.90 |
| Cont. case violation threshold | 1.00 |
| Low voltage warning threshold | 1.05 |
| High voltage warning threshold | 1.15 |
| Low voltage violation threshold | 0.90 |
| High voltage violation threshold | 1.10 |
| Do not consider any contingency below this voltage (KV) | 100.0 |

Figure 2.13: Contingency Analysis options in the platform

Figure 2.14 shows the tabular results of the CA returned by the platform and rendered by the SPA. The results are sorted by line loading percentage. The user also has the option to not consider a contingency by unchecking the “Active” checkbox corresponding to each contingency.

I used Tableau visualization software to create a better visualization of the contingencies using the contingency report the platform makes available for download. Figure 2.15 shows the resulting contingency matrix diagram.

The platform parses the contingency analysis results in an interactive table. The user can search and filter monitor and contingency branches using the results table provided by the platform. She can also sort the contingencies based on any column. It should be emphasized that in this context, “contingency” is the branch that goes out of service and causes the “monitor” branch to overload. Note that a simple contingency could cause multiple monitor branches to go into warning or violation state.

| Active ^{↑↓} | Status ^{↑↓} | Case ^{↑↓} | Contingency Branch ^{↑↓} | Monitored Branch ^{↑↓} | Pre Cont. Flow ^{↑↓} | Post Cont. Flow ^{↑↓} | Line Loading % ^{↑↓} | Severity ^{↑↓} |
|-------------------------------------|----------------------|--------------------|----------------------------------|--------------------------------|------------------------------|-------------------------------|------------------------------|------------------------|
| <input checked="" type="checkbox"/> | Current | Post Cont. | tx2-2157-157-1 | In-2121-2380-1 | 16.87 | 84.86 | 105.42 | Violation |
| <input checked="" type="checkbox"/> | Current | Post Cont. | In-894-1514-1 | In-939-1416-1 | 113.6 | 162.49 | 100.93 | Violation |
| <input checked="" type="checkbox"/> | Current | Post Cont. | In-32-36-1 | tx2-540-23-1 | 77.21 | 184 | 100 | Warning |
| <input checked="" type="checkbox"/> | Current | Post Cont. | In-67-138-1 | In-66-152-1 | 227.22 | 530.2 | 99.36 | Warning |
| <input checked="" type="checkbox"/> | Current | Post Cont. | In-294-346-1 | In-344-346-1 | 79.72 | 102.53 | 99.07 | Warning |

Figure 2.14: Contingency analysis results table in the platform

2.2.5 Security Constrained Economic Dispatch (SCED)

SCED solves an optimization problem to determine the set points for network generators so they operate with least cost and also satisfy all the reliability constraints obtained from running RTCA. SCED module has been developed by a former team member and PhD candidate Xingpeng Li. By assuming a simple DC model for the network, SCED simplifies the problem to a linear programming problem. SCED uses Gurobi [18] optimization software to solve the formulated linear programming (LP) problem. The constraints for the LP are line flow limits, generator production limits, generation and consumption equality constraints, and contingency constraints obtained from RTCA. A detailed formulation and explanation of the optimization problem is given in [19]. Figure 2.16 shows the general architecture of the SCED package. Note that SCED uses the whole RTCA package (container in c4 vocabulary). SCED also needs network sensitivity factors [20]. OpenPA Tools package provides the necessary functions for sensitivity factor calculations.

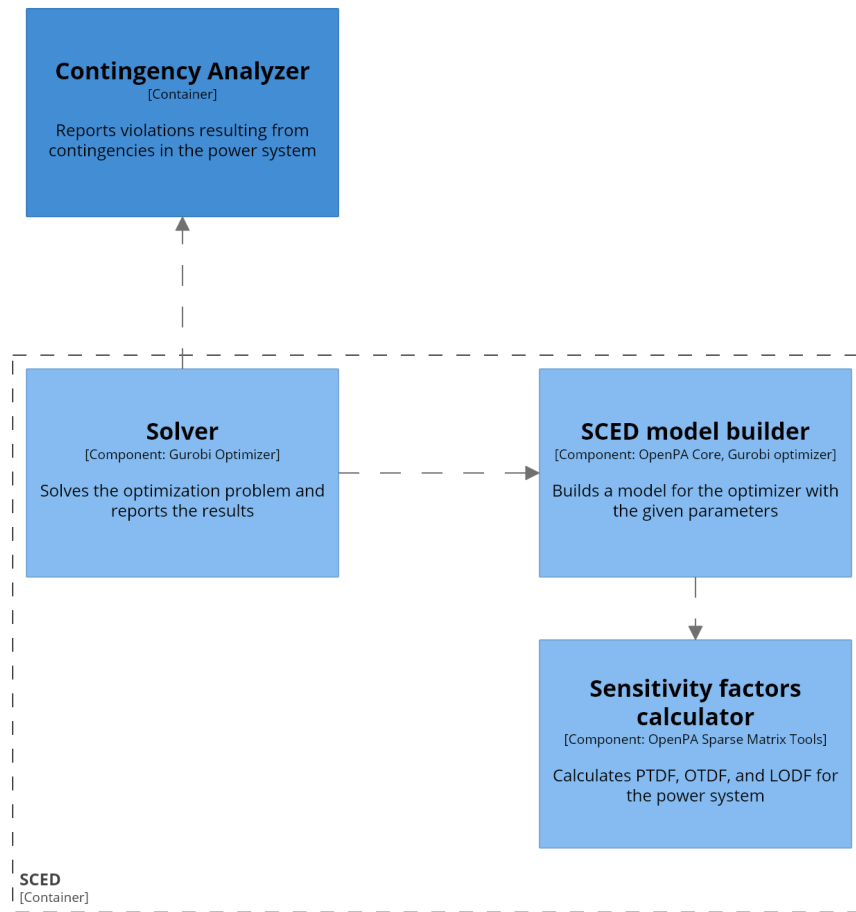


Figure 2.16: SCED Components Diagram

Figure 2.17: SCED Options Form 1

Figure 2.18: SCED Options Form 2

The user can also choose to render the nodes based on the change in generator dispatch calculated by SCED. Figure 2.20 shows how the generators dispatch points changes after a SCED.

It should be noted that at this point, SCED dispatch updates are applied automatically to the generators and there is no operator review process in the platform. The user can only see information regarding the change. But it is assumed that all the changes are accepted and applied to the generators, hence, making the next loop start with a new dispatch point.

SCED Results

Show entries Search:

| id | Dispatch Difference (MW) |
|------------|--------------------------|
| gen-127-1 | 162.64 |
| gen-44-1 | 157.51 |
| gen-17-1 | 151.78 |
| gen-1538-1 | 145.01 |
| gen-105-1 | 126.03 |
| gen-104-1 | 85 |
| gen-85-1 | 80.01 |

Figure 2.19: SCED Results Table

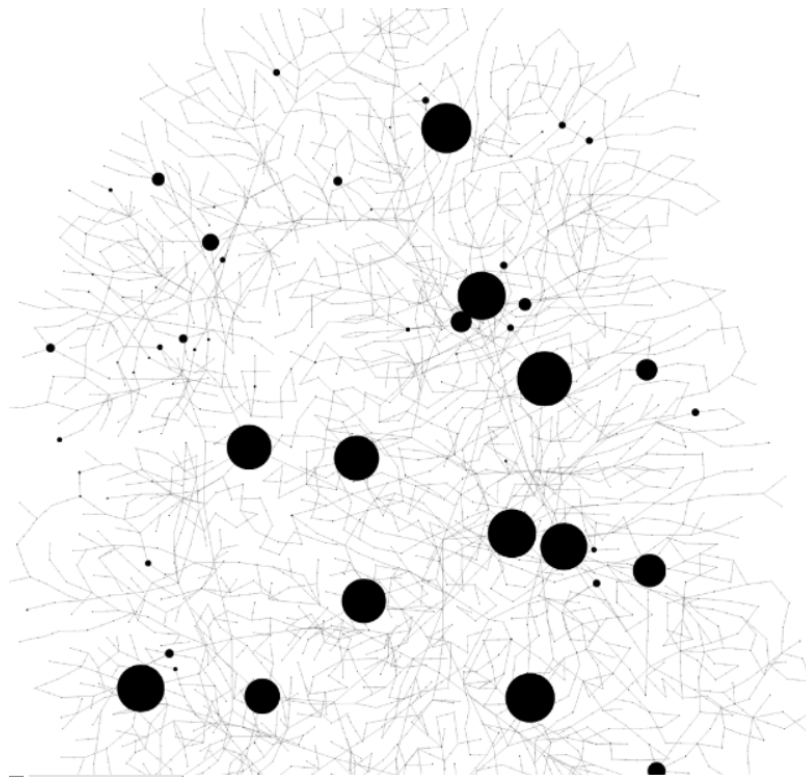


Figure 2.20: Dispatch differences after SCED: the circles with larger areas denote larger change after re-dispatching according to the SCED solution.

2.2.6 *Anomalous Data Detector*

This detector has been contributed by the Arizona State University team member and PhD candidate Andrea Pinceti. It uses a machine learning techniques and a set of historical loads to detect an anomalous data point. The code for the detector has been written entirely in MATLAB programming language. The platform gets model load data from OpenPA and uses the Java MATLAB engine to start a MATLAB process and pass the load vector to the MATLAB function which in turn detects if the load vector is anomalous or not. As figure 2.1 shows, at this point, the system does not do anything if an anomalous load is detected. In other words, this detector only has only advisory role and will not change the system state in any way. The MATLAB function returns all detection parameters to the calling module. These data can be a basis for later decision making and operator review.

As mentioned, this module is directly executed inside a MATLAB process. However, using a MATLAB-Java connector, the platform starts the MATLAB process in the background and maintains it in the memory. Every time this module runs the MATLAB function, the platform passes the data between the main Java application to the MATLAB process and retrieves the results.

2.3 Web Application

To glue together all the analytical libraries, the visual front end, and the DES event loop, a web application has been created in Scala [21] programming language using client-server application model. Scala is a java-like programming language which unifies object-oriented and functional programming styles [21]. The reason for using Scala is its brevity, powerful collection libraries, and the ability to write semantically clear code. To develop the web application, a web application framework called **Play** (www.playframework.com), which supports both Java and Scala programming languages, was chosen.

Figure 2.21 shows an overview of the components in the web application.

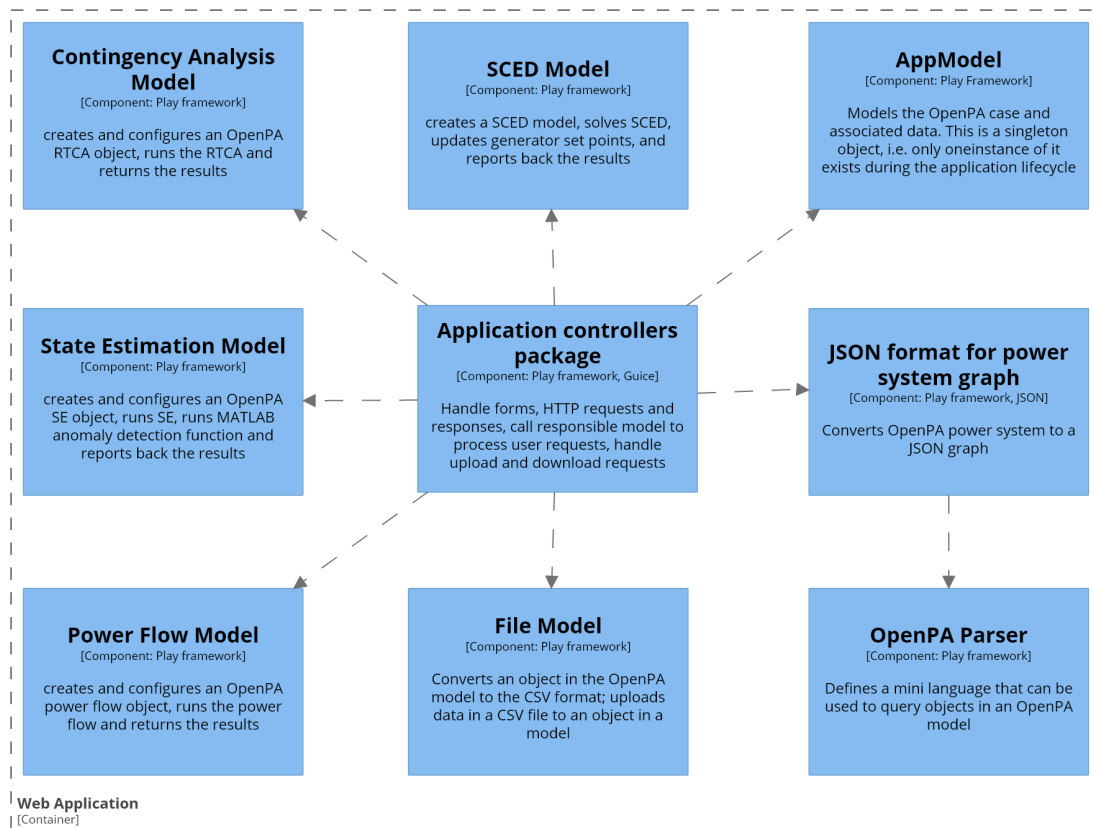


Figure 2.21: Web Application Components

Web application follows the Model-View-Controller (MVC) paradigm:

1. Models: responsible for sending and receiving data from the underlying case managed by OpenPA
2. Controllers: responsible for routing the requests to the appropriate model and send back the results to the view
3. Views: responsible for result presentation

The goal of using MVC is mainly separation of concerns. This architecture allows virtually independent development of each component without severely affecting other components. For example, more sophisticated simulation models can be used in the back-end while the front-end application can also change completely for educational or industrial purposes. As long as the messaging protocol between the two remains consistent, it is not hard to update or even completely re-write the front-end or the back-end without breaking the other one.

The web application also uses dependency injection (DI) to make the dependencies of each component explicit. Google Guice framework was chosen to handle dependency injections [22]. Each modules' dependencies are declared at the beginning and are explicitly injected by our DI framework. DI also handles the life cycle of both models and controllers. For example, our `AppModel` (2.21) module keeps the power system state. By declaring this module as `Singleton`, DI framework ensures that this object is maintained in memory as long as the application is running. This is in contrast to declare a static object or manually maintaining instances and both increases efficiency and makes code maintenance easier.

The web application also follows REST [23] principles. Every resource (a functionality or piece of data in the application) is assigned to a unique path which enables

retrieval through REST actions (GET or POST). Moreover, the application operates in a stateless manner: each request from the client is handled independently from other requests and the application does not store any state between requests. Of course, a power system has a certain state at any point in time. As mentioned above, the responsibility of managing state has been delegated to the dependency injection framework.

An example clarify things with an example. When the user requests a power flow, the web application first receives this request via a controller. It then instantiates a `PfModel` object which creates and configures a `FDPowerFlow` object which in turn is responsible for the actual power flow analysis. After power flow analysis is over, the underlying case is updated by calling `updateResults()` method from OpenPA core. This method directly changes the data structures managed by `AppModel`. Since `AppModel` is a singleton is always maintained in the memory, the change in state will be permanent.

2.4 OpenPA Parser

I have written an OpenPA parser in Scala programming language. OpenPA parser is a simple mini-language that the programmer can use to query OpenPA core's in-memory case model and obtain information from it. Table 2.1 summarizes the language keywords and operators. The **Bus** and **Branch** sections of the table define the properties the user can query from the power system. Each column in the section defines the property type. For example, the property `voltagelevel` is a numerical property while the property `id` is of the type string.

For example, to get all buses with voltage level above 300KV, the programmer can use the following code:

```
voltagelevel > 300
```

| Bus properties | | | Branch properties | | | Operators | | |
|----------------|---------|------------|-------------------|---------|-------------|-----------|----------|---------|
| numerical | text | boolean | numerical | text | boolean | numerical | text | boolean |
| voltagelevel | id | pinj_telem | fromp | id | fromp_telem | >= | eq | is |
| pinj | station | qinj_telem | pfrom | frombus | top_telem | <= | contains | and |
| qinj | name | pinj_bdd | fp | tobus | fromq_telem | == | | or |
| vmpu | owner | qinj_bdd | pf | name | toq_telem | != | | |
| vm | area | | fromq | index | fromp_bdd | < | | |
| va | index | | qfrom | | top_bdd | > | | |
| active_load | genid | | fq | | fromq_bdd | | | |
| active_gen | | | qf | | toq_bdd | | | |
| | | | X | | | | | |
| | | | R | | | | | |
| | | | flow | | | | | |
| | | | ltrating | | | | | |

Table 2.1: OpenPA parser mini-language. The table has three sections. Each section has three columns which define the property types.

This command will programmatically be translated to (Scala notation):

```
model.getBuses.filter(bus => bus.getVoltageLevel.getBaseKV > 300)
```

To retrieve a line with a known id, the following code can be used:

```
id eq 'ln-123-124'
```

For which the programming instructions would be:

```
model.getLines().getByID("ln-123-124")
```

Figure 2.22 shows the partial abstract syntax tree of the parser. Operands will be translated to the appropriate function based on the operand type and each operand has its own parser which is shown in figure 2.23.

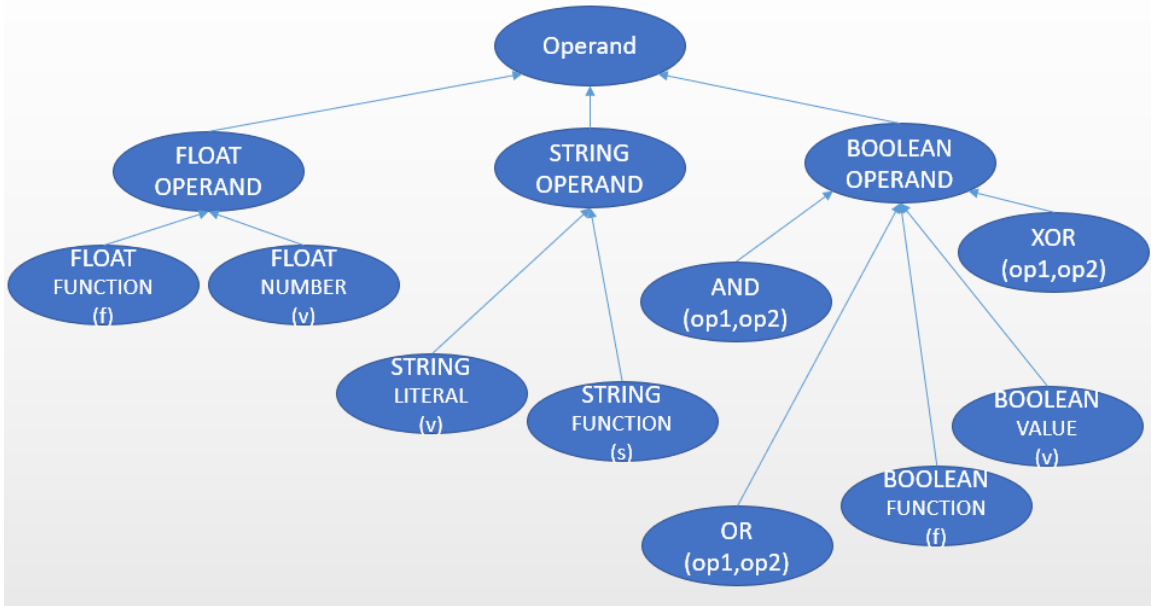


Figure 2.22: OpenPA Parser Abstract Syntax Tree

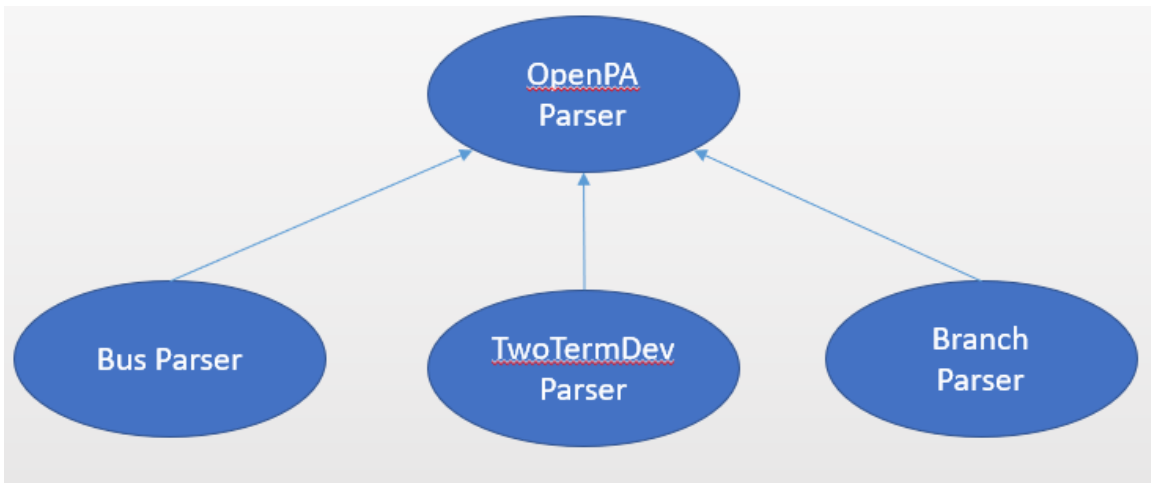


Figure 2.23: OpenPA Parser Inheritance Hierarchy

This mini-language provides a simple way to acquire data from the model using human-like language. This can be useful when trying to query the model remotely. The language can also be extended to enable it to send commands to the model as well as retrieving data from it. OpenPA parser is used in the single page application

(SPA) to create system graph visualizations based on user criteria.

2.5 Single Page Application

The single page application (SPA) has been written in JavaScript and HTML programming languages. It uses Bootstrap, JQuery, and CytoscapeJS [24] libraries. It is responsible for visualizing and delivering reports to the user as well as receiving user commands which can be either a request for normal analysis or some form of data injection request. JQuery is a well-known general-purpose JavaScript library. CytoscapeJS has been developed by bioinformatics researchers, although it is a general-purpose visualization library with a lot of out-of-the-box graph manipulation capabilities. The main purpose of the SPA is to facilitate working with various OpenPA packages and libraries through a visual interface. I have Incorporated almost all the options available through direct writing of code in the SPA. The user can get almost everything OpenPA applications have to offer without writing code. This makes the SPA an ideal tool to use in classrooms and presentations. The following is a list of features currently supported by the SPA:

- The user can configure any study with all the available options through a graphical user interface. After running the study, the results can be examined on the network graph or by downloading a summary report generated by the server.
- Ability to filter the elements based on the OpenPA parser mini language. The user can choose to assign element colors, node sizes, and line widths according to the results of the filtering operations.
- The user can change node sizes or branch widths based on definable criteria, such as: generation, load, dispatch changes after SCED, line flows, and line limits.

- The visualized graph is interactive. The user can click on nodes and links to see corresponding system data; the user can also drag nodes to another location and keep the layout by saving it on the server.
- The SPA allows the user to use CytoscapeJS layout algorithms such as Cose to obtain an initial rendering if there are no geographical data available for the system.
- The SPA allows the user to view the simulation loop and the current stage of simulation with a graphical diagram. The user can also interact with the diagram directly.

The SPA also has its own modular architecture. Figure 2.24 shows the general architecture of the SPA. The graph component and the form component reside in their own independent modules and can only be fetched via calls from the main event handler application.

In this chapter, I went into detail about different building blocks of the simulation platform. Although I mainly discussed IncSys's power analysis libraries and the options they provide for the programmer, I also showed how those libraries can be used to create a modular discrete event simulation event loop based on a client-server model.

In the next chapter, I show how the platform can be used to simulate FDI attacks and analyze the consequences. I also show how the user can make use of the platform's user interface to create sophisticated visualizations to help clarify the attack both for expert and non-expert users.

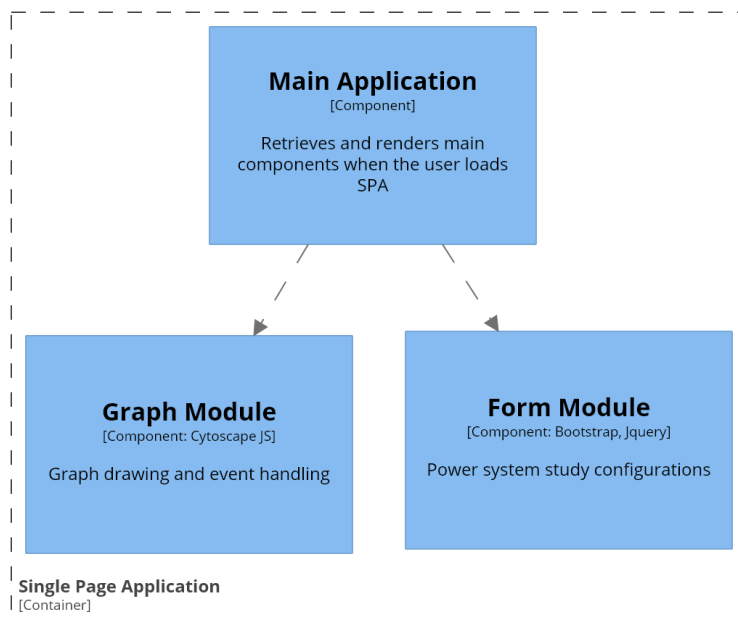


Figure 2.24: SPA Components

Chapter 3

CASE STUDIES

In this chapter, I go over two different case studies carried out using the EMS platform and present the results and discuss how the user can use the tools available in the platform to design, analyze, and showcase two successful FDI attacks. As discussed in chapter 1, the attack described in [8] is simulated. Rather than going into the theoretical notions, the goal is to test the attack using the simulation platform and analyze the consequences. I also present an overview and discuss possible future work regarding the simulation platform.

3.1 General Attack Mechanism

Figure 3.1 shows the simulation loop the platform presents to the user. This diagram is a simplified version of 2.1. The user can interact with the diagram and select the simulation step they desire. The user can run the operations in the general EMS loop (figure 2.1) using the following recipe:

1. The user runs a power flow study to simulate SCADA measurements across the system.
2. The user then changes hats and acts as an operator who runs an state estimation study in the control room. FDI attack injection can also be carried out at this stage. The “Bad Data Detection” box and the “Anomalous Data Detection” box will use the results of SE to check for bad or anomalous data according to their corresponding algorithms. The user cannot interact with these boxes.
3. The next step in the simulation is the CA box. Again, the user can fully

configure the CA in its corresponding tab. Clicking on this box will request and ultimately execute a contingency analysis using OpenPA RTCA.

4. Finally, the user can run a SCED analysis to set the new dispatch points for all the active generators in the system.

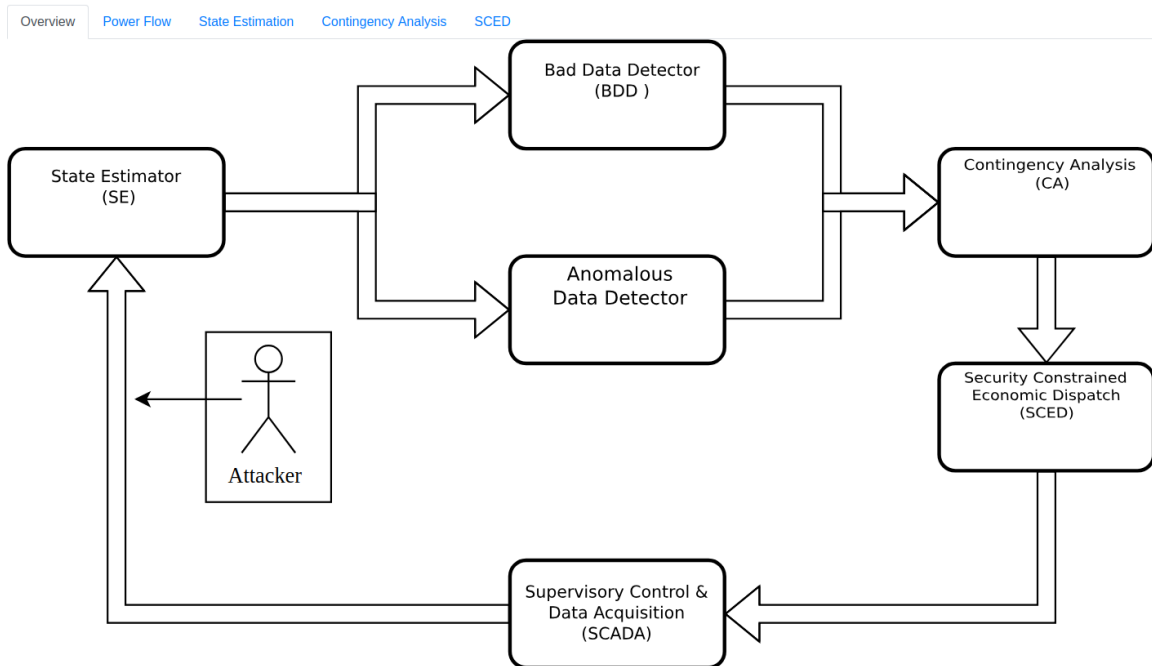


Figure 3.1: Simplified EMS loop diagram in the simulator

While the user can potentially inject data at any point in this process to simulate attack or an anomalous event, to carry out an FDI attack, the user should inject the false measurements during the state estimation phase of the EMS. Currently, the platform does not have the capability to calculate an attack vector directly. The attack vector and the resulting measurements are calculated offline (although the offline calculation relies on power flow and state estimation reports from the platform). The user can then inject the prepared measurements during state estimation by clicking on the “Attacker” box.

3.2 Attack Against Polish System

The attack described in [8] is tested with the following modification: instead of misleading operators to create line overloads in the system as [8] suggests, this attack will create false contingencies in the system. This means an unforeseen overload will only happen if a contingency occurs in the system. In other words, this attack compromises the $N - 1$ -secure state of the network (explained in chapter one) that the operating standards require maintaining.

The attack process is similar to the normal EMS loop operation except that the user may upload pre-calculated false measurement data after state estimation. The process has become simpler by adding a box to the simulator that automatically injects the pre-calculated false measurements to the model. The user can even upload multiple false measurements files to the server and choose to inject any of them. In any case, the user can always use the platform's "Upload Data" feature to inject measurements.

As noted in chapter 2, the bad data detector should be able to sense relatively small changes in the data. However, the false measurements in this case have been calculated using a bi-level optimization problem and are specifically designed to pass the bad data detector. This fact is proven after the BDD does not raise any alarm over the false data. However, the false measurements cannot pass the anomalous data detector since that detector uses historical load data.

The working assumption is that the operators are unaware of a false data injection attack and continue to believe the data out of the state estimator are authentic. These measurement data results in radically different bus injection values than the values in the real world. This change in injection is construed as a change in loads as generators are assumed to be under operator control and having low probability of

sudden changes. Unaware of the attack, the operator continues to operate the cyber system by running RTCA and SCED and applying the SCED dispatch setpoints to all the generators in the system. This is a crucial point in the attack process since the cyber and the physical systems now behave in completely different manners. To see this difference, the user can first check the cyber system by running another round of power flow (to simulate SCADA measurements), state estimation, and RTCA. The reported contingencies are almost the same as before the attack and give no reason for suspicion. The user may save these results either by downloading them or clicking “Save Contingencies” button in the “Contingency Analysis” tab which will save a copy of the contingency map in memory.

To analyze the physical system with the dispatch resulting from the false measurements, the user can simply click the “Reset Loads” button shown in the main simulation diagram. Clicking this button effectively switches the system from cyber to physical. Note that the dispatch is still the same from the previous round of EMS meaning that the dispatch is induced by the false data. After resetting the loads, a power flow and a contingency analysis are executed. There is no need for state estimation since it is the physical system that is being simulated.

At this point the attack is complete: two RTCA studies have been executed, one in the cyber world and one in the simulated physical world. Normally it is expected that the RTCA reports to be very similar. However, figure 3.2 tells a different story.

In figure 3.2, each dot represents a contingency, the horizontal represents the line loadings in the cyber world, and the vertical axis stands for the line loadings in the physical world. This diagram. In other words, the horizontal axis is about the contingencies the operator sees, and the vertical axis represents the contingencies in the physical world. If the cyber and physical world behaved the same way -as expected from a normally functioning EMS - there should have been no discrepancy. On the

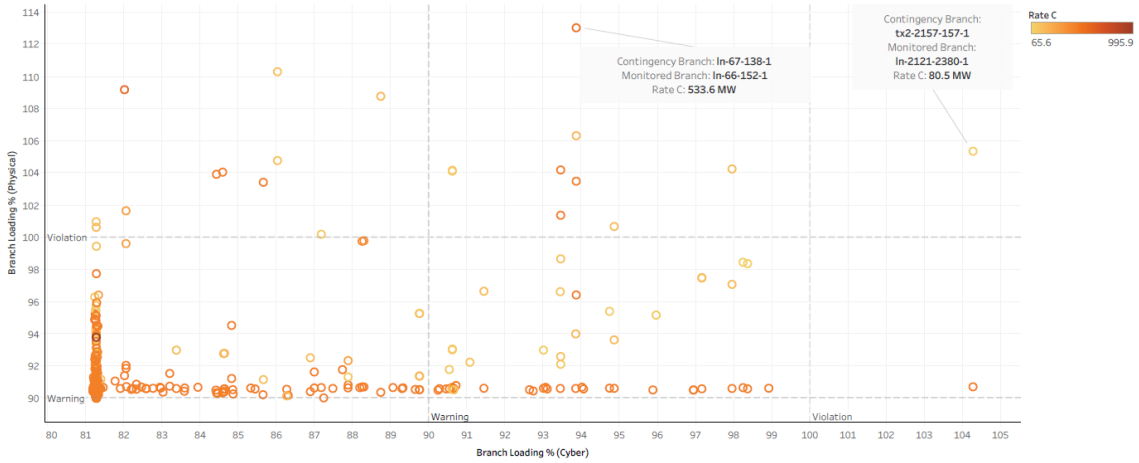


Figure 3.2: Attack consequences diagram for the Polish system. The guidelines denote the thresholds for various alarms.

contrary, figure 3.2 shows that some very serious contingencies exist in the physical world that the operator is not able to see in the physical world. This means despite consuming considerable time and energy, the system would not operate without loss of lines in a contingency event, completely defeating the purpose of RTCA and SCED in the first place in addition to possible severe consequences for the network.

Plotting the diagram in figure 3.2 would not have been possible without the features available from OpenPA RTCA module. To obtain hidden contingencies, I ran a RTCA with lower threshold values. This made it possible to view branches that would not normally show up. For this plot, I used RTCA threshold values as low as 80%.

The worst consequence of the attack is a 13% overload in the physical world which shows only as a WARNING (94% loading) level alarm in the cyber world. Figure 3.3 shows these two branches marked on a graph of the Polish system. If the 400KV line goes out of service, the 220KV will be forced to pick up more load and that will cause thermal overload violations. The attack has deceived the operators to dispatch the

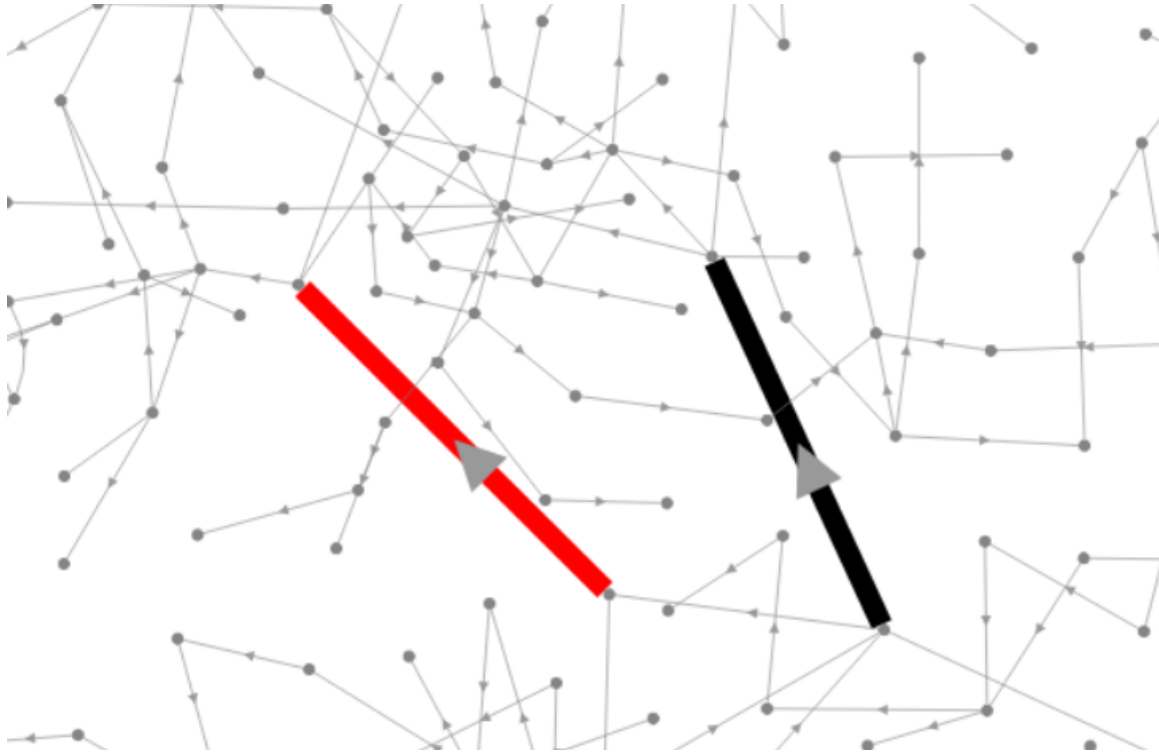


Figure 3.3: The worst overload in the physical world in the Polish system attack. The branch marked in black (ln-67-138-1: 400KV line) is the contingency branch and the branch marked in red (ln-66-152-1: 220KV line) is the monitor branch.

generators in a way that increases the severity of such overload; it also hides the true contingency from the operator eyes.

3.3 Attack Against Texas Synthetic System

Our team has conducted another study using the ACTIVSg (Texas synthetic) system [17]. Due to design issues, the team decided to derate a few lines to make SCED feasible for this system. The list of lines to be derated are available from the SCED options 2 tab. The user can also experiment with further deratings if she wishes so.

The attack process is similar to the Polish system. The attack consequences diagram (obtained using a similar method to the Polish case) is shown in 3.4.

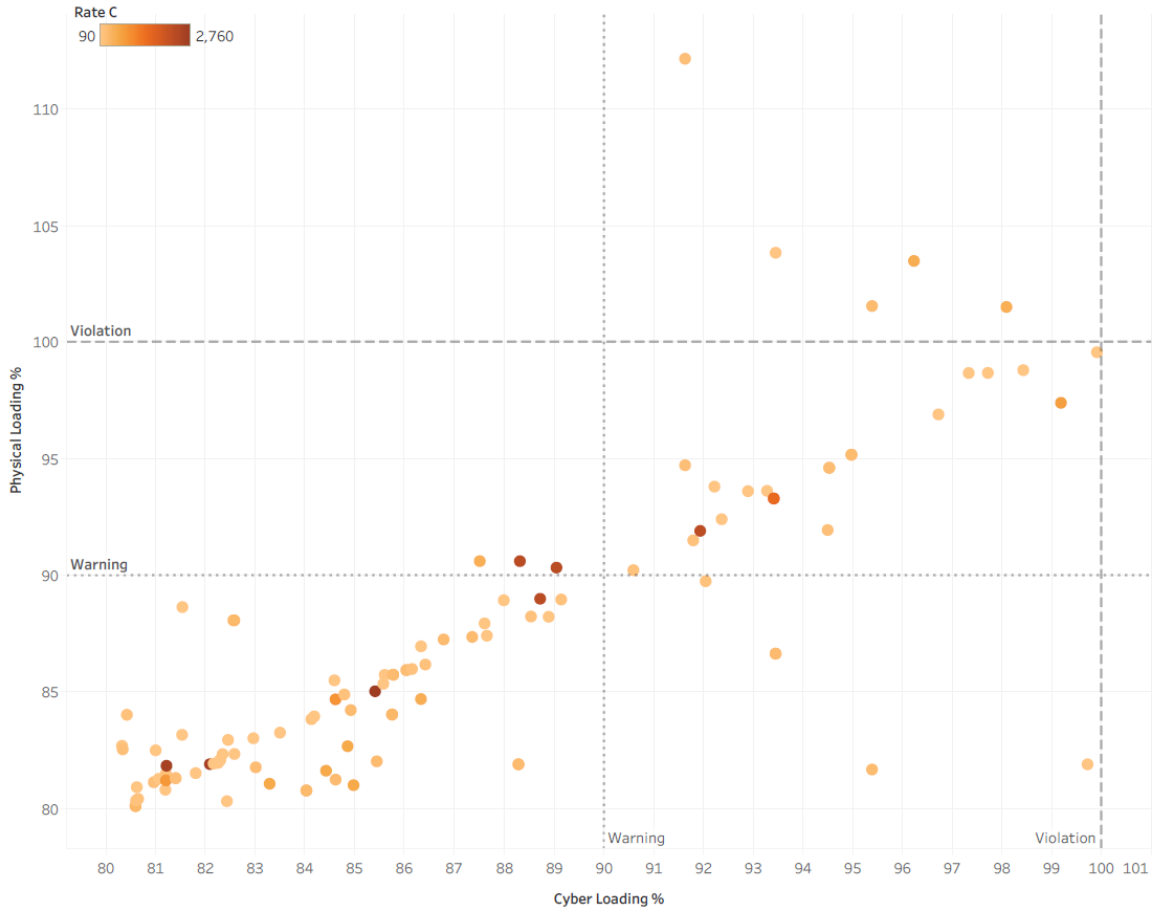


Figure 3.4: Texas System Attack Consequences

Unlike the Polish case, most contingencies in both cyber and physical world seem to have the same overload values. But the attack has succeeded in creating a very severe hidden contingency. Figure 3.5 shows the geographical locations of the lines. They are in the same approximate geographical location (Wenatchee, TX).

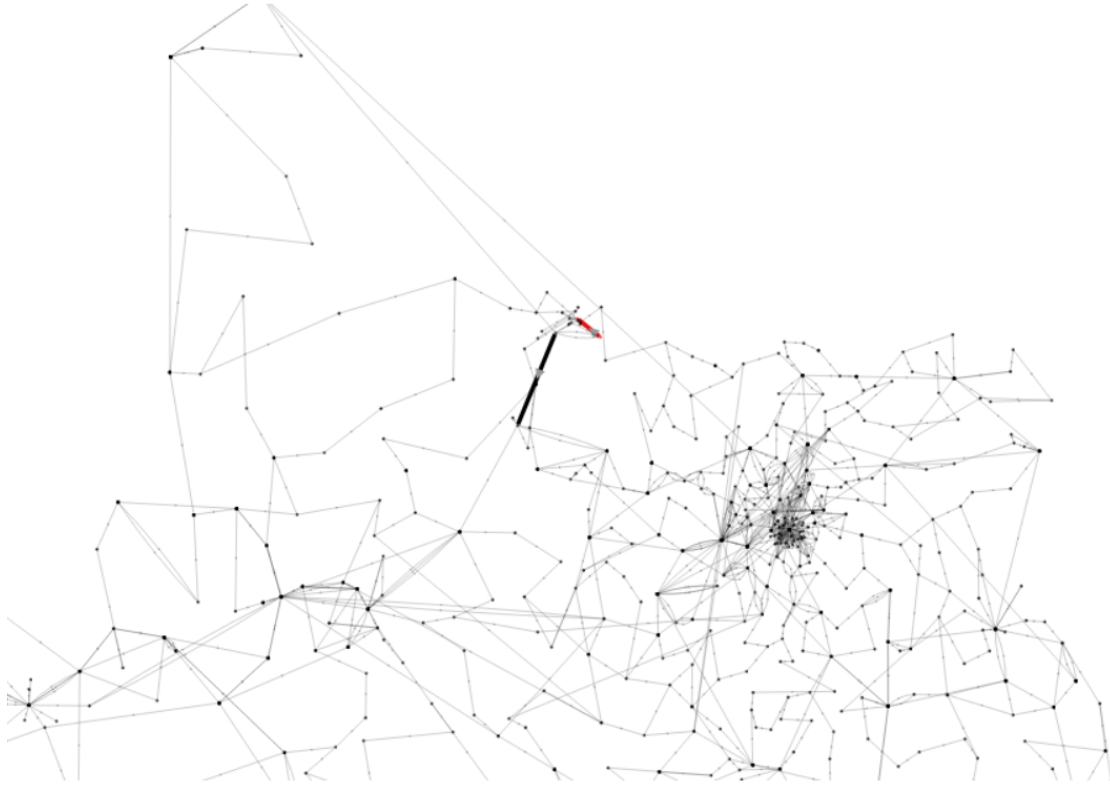


Figure 3.5: Geographical location of the lines affected most by the Texas system attack. Both lines are located in Wenatchee, Texas. The line marked in black is a 500KV line; when it goes out of service the 161KV line marked in red will overload

3.4 Conclusion and Future Work

I reviewed power system operations and planning in smart grid concepts, justified the need for power system simulation, then proposed a software architecture for implementing the simulator using an open source power system analysis library created by our industry partner. I also explained in detail various parts of those libraries and showed how the programmer can extend them to her benefit.

I also coded and deployed the designed simulator in a modular fashion. This platform removes the need for coding and compiling Java programs as it makes all the options in OpenPA PF, SE, and RTCA available to the user via a graphical

user interface. The platform is also a DES event loop with the extra capability of simulating FDI attacks. Moreover, it provides power system visualization and reporting.

Finally, I used our platform to examine the consequences of FDI attacks against two synthetic cases. I used the platform to simulate the process in a fast and efficient manner, generate better reports, and create various visualizations to help both the expert and non-expert user understand the attack consequences.

The platform has the potential for extension both in the front end and in the back end. I mention a few of such improvements here:

In the front end, better visualization techniques can be used for power system graph visualization. This can be very useful especially if the platform is to be used in classrooms and presentations. Building further upon the REST architecture, the platform can be transformed into a service that can send the required data to any front end application.

In the back end, the platform can be extended by being connected to a database system. This will allow efficient storage and retrieval of the historical data.

It should be noted that currently the platform provides only limited operator behavior simulation. SE and SCED results are automatically uploaded to the underlying model without any review or action from the operator and contingencies are filtered only by rudimentary criteria. Introducing AI based operator behavior simulation into the DES event loop can lead to more realistic scenarios revolving around operator decisions.

REFERENCES

- [1] EIA, “Electric Power Annual 2017,” Tech. Rep. October, EIA, 2018.
- [2] S. Borlase, *Smart Grids : Infrastructure, Technology, and Solutions*. Electric Power and Energy Engineering, Boca Raton, FL: CRC Press, 2012.
- [3] C. Greer, D. A. Wollman, D. E. Prochaska, P. A. Boynton, J. A. Mazer, C. T. Nguyen, G. J. FitzPatrick, T. L. Nelson, G. H. Koepke, A. R. Hefner Jr, V. Y. Pillitteri, T. L. Brewer, N. T. Golmie, D. H. Su, A. C. Eustis, D. G. Holmberg, and S. T. Bushby, “NIST Framework and Roadmap for Smart Grid Interoperability Standards, Release 3.0,” tech. rep., National Institute of Standards and Technology, Gaithersburg, MD, oct 2014.
- [4] S. D. Anton, D. Fraunholz, C. Lipps, F. Pohl, M. Zimmermann, and H. D. Schotten, “Two decades of SCADA exploitation: A brief history,” *2017 IEEE Conference on Applications, Information and Network Security, AINS 2017*, vol. 2018-January, pp. 1–8, 2018.
- [5] R. E. Shannon, *Systems simulation: the art and science*. Prentice-Hall, 1975.
- [6] F. Schloegl, S. Rohjans, S. Lehnhoff, J. Velasquez, C. Steinbrink, and P. Palensky, “Towards a classification scheme for co-simulation approaches in energy systems,” *Proceedings - 2015 International Symposium on Smart Electric Distribution Systems and Technologies, EDST 2015*, pp. 516–521, 2015.
- [7] B. Chen, S. Mashayekh, K. L. Butler-Purry, and D. Kundur, “Impact of cyber attacks on transient stability of smart grids with voltage support devices,” *IEEE Power and Energy Society General Meeting*, pp. 1–5, 2013.
- [8] J. Liang, L. Sankar, O. Kosut, and K. Hedman, “Consequences of False Data Injection on Power System State Estimation,” Master’s thesis, Arizona State University, 2015.
- [9] R. D. Zimmerman, C. E. Murillo-sánchez, R. J. Thomas, L. Fellow, and A. M. Atpower, “MATPOWER : Steady-State Operations , Systems Research and Education,” *IEEE Transactions on Power Systems*, vol. 26, no. 1, pp. 12–19, 2011.
- [10] J. Gosling, B. Joy, G. Steele, G. Bracha, A. Buckley, and D. Smith, *The Java ® Language Specification Java SE 12 Edition*. Oracle America, Inc, java se 12 ed., 2019.
- [11] C. Mosier, “OpenPA,” 2013. Available at github.com/powerdata/openpa-private.
- [12] R. Podmore, *Digital computer analysis of power system networks*. PhD thesis, University of Canterbury, Christchurch, New Zealand, 1972.
- [13] R. Klurnp, “Understanding Object-Oriented Programming Concepts,” in *2001 Power Engineering Society Summer Meeting*, vol. 1070, pp. 1070–1074, 2001.

- [14] P. S. Bodger, *Fast decoupled AC and AC/DC loadflows*. PhD thesis, University of Canterbury, Christchurch, New Zealand, 1977.
- [15] S. G. B. Wood Allen J., Wollenberg Bruce F., “9.3.1 introduction,” 2014.
- [16] T. Davis and Y. Hu, “The university of Florida sparse matrix collection,” *ACM Transactions on Mathematical Software (TOMS)*, vol. 38, no. 1, pp. 1–25, 2011.
- [17] A. B. Birchfield, S. Member, T. Xu, S. Member, K. M. Gegner, S. Member, K. S. Shetye, and T. J. Overbye, “Grid Structural Characteristics as Validation Criteria for Synthetic Networks,” *IEEE Transactions on Power Systems*, vol. 32, no. 4, pp. 3258–3265, 2017.
- [18] L. Gurobi Optimization, “Gurobi optimizer reference manual,” 2018.
- [19] X. Li and K. W. Hedman, “Enhanced Energy Management System with Corrective Transmission Switching Strategy-Part I: Methodology,” *arXiv preprint arXiv:1810.05940*, 2018.
- [20] G. B. Wood Allen J., Wollenberg Bruce F Sheble., “Calculation of Network Sensitivity Factors,” 2014.
- [21] M. Odersky and al., “An Overview of the Scala Programming Language,” Tech. Rep. IC/2004/64, EPFL, Lausanne, Switzerland, 2004.
- [22] R. Vanbrabant, *Google Guice*. APress, 2008.
- [23] R. T. Fielding, *Architectural styles and the design of network-based software architectures*, vol. 7. University of California, Irvine Irvine, USA, 2000.
- [24] M. Franz, C. T. Lopes, G. Huck, Y. Dong, O. Sumer, and G. D. Bader, “Cytoscape.js: A graph theory library for visualisation and analysis,” *Bioinformatics*, vol. 32, no. 2, pp. 309–311, 2015.