Local Electricity Market (LEM) Simulator Reference Manual

January 2023

Contents

1 Overview

This draft manual provides information on how to build and run the Local Electricity Market (LEM) simulation model described in Dedrick, Perrin, Sabaghian and Wilcoxen, "Assessing cyber attacks on local electricity markets using simulation analysis: Impacts and possible mitigations," *Sustainable Energy, Grids and Networks*, 2023.

2 Installation

The model depends on two external libraries:

commons-csv-1.4.jar
: Commons CSV reads and writes files in variations of comma separated value (CSV) format. More details can be found on the project official website.

mason.19.jar
: The LEM simulator uses the MASON discrete event simulation framework to manage steps happening in the dynamic distributed market simulator. More details can be found on the project official website.

To build the simulator:

If you have the build tool "make", you can use the command line to enter the directory "src" and then execute the command "make."

If you do not have any tool that is able to process "makefile", execute the following commands manually:

1. Launch the command line and change the working directory to the directory "src".
2. Execute the following command to compile the java code:
```
javac –cp “../ lib /∗;.” $∗.java
```
3. Generate the java archive file with the following command:
```
jar cvfm market.jar manifest.txt ∗. class
```
4. After running the commands above, you can obtain the archive file "market.jar" in the "src" directory.

3 Running

3.1 Run the Simulation

If you have built the simulator successfully, you should be able to find a Java archive file "market.jar" in the directory "src". Then you can follow the steps below to run a simulation:

1. Launch the command line and change the working directory to the directory "run".

2. Execute the following command:

```
java –jar ../src/market.jar <run_configuration_file>
```

More details about the run configuration files can be found in the next section.

3.2 Run Configuration Files

Run configuration files provides environment configurations for the simulator, describing environmental information that will be used in the simulation.

Table 1: Run configuration options

Option	Meaning	Type	Default
netmap	Path of the grid agent configuration file.	Required	
virtualmap	Path of the virtual agent configuration file.	Optional	none
draws	Path of the draw file.	Required	
history	Path of the historical output file.	Optional	none
bids	Path of the bid file.	Optional	none
transcost	Transmission cost between nodes.	Optional	1
transcap	Maximum transmission between nodes.	Optional	2500
seed	Seed for random number generator.	Optional	none
debug	Debugging flag (0, 1, default: 0).	Optional	0
numpop	Number of populations to draw.	Optional	10
dosprop	Proportion of messages lost in DOS attacks.	Optional	0,1,5,10

Note that if you set the "transcost" and "transcap" in the run configuration file, then their values will be applied to all the grid agents in the market. Cost and capacity settings for individual agent in the file "netmap.csv" will be ignored.

3.2.1 Example

The following example is from the configuration file "c2k1800.txt". This file is used for simulation without any adversary. In this case, you do not need specify "virtualmap", "history", and "bids" files since these files are only used by virtual agents.

```
netmap    : netmap.csv
draws     : testdraw.csv
transcost : 2
transcap  : 1800
seed      : 123456
debug     : 1
```

This configuration file indicates that “netmap.csv” and “testdraw.csv” in the current directory will be used as grid agent configuration and draw file, respectively. Transmission cost between two nodes will be 2 and maximum transmission between two nodes will be 1800. “123456” will work as seed for random number generator. Debugging messages will also be enabled in this simulation.

3.2.2 Example

The following example is from the configuration file “c2k1800d.txt”. This file is used for simulation with adversary “AdvDarth”. Note that you have to specify “virtualmap”, “history”, and “bids” files for the adversary “Darth”.

```
netmap     : netmapd.csv
virtualmap : virtualmapd.csv
draws      : testdraw.csv
history    : c2k1800_out.csv
bids       : c2k1800_net.csv
transcost  : 2
transcap   : 1800
seed       : 123456
debug      : 1
```

4 Input

4.1 File netmap.csv

"Netmap" file provides grid agent configuration for the simulator, describing details of each grid agent in the electricity market.

Table 2: Netmap file columns explanation

Column	Meaning
id	Unique identifier of the grid agent.
type	Grid agent's role in the electricity market: 1 = Level 1 market, top level; 2 = Level 2 market; 3 = Trader.
sd type	Grid agent's type as supplier (S) or consumer (D).
up id	ID of the upstreaming node that the current node connects to.
channel	Communication channel used by the current node.
cost	Transmission cost.
cap	Transmission capacity.
security	Security level (0-100) of the current grid agent.

4.1.1 Example

The following example is the configuration of node 1:

Table 3: Example configuration of node 1 in netmap.csv

Line in file:

```
1,3,D,201,type1,1,2000,100
```

Column	Value	Meaning
id	1	This grid agent's id is 1.
type	3	This grid agent is a trader working on the level 3.
sd type	D	This grid agent is a consumer.
up id	201	This grid agent connects to upstreaming node 201.
channel	type1	This grid agent is using type1 channel.
cost	1	The transmission cost on this wire is 1.
cap	2000	The transmission capacity of this wire is 2000.
security	100	This grid agent's security level is 100.

4.2 File virtualmap.csv

"Virtualmap" file provides virtual agent configuration for the simulator, describing details of each virtual agent in the electricity market.

Table 4: Virtualmap file columns explanation, part 1

Column	Meaning
id	Unique identifier of virtual agents.
type	Virtual agent type, including AdvAdam, AdvBeth, AdvDarth, AdvElvira, AdvFaust.
configuration	Configuration parameters for virtual agent's behavior. Please see the table below for additional details.
channel	Communication channels the virtual agent has access to.
agent	ID of grid agents that virtual agent has access to.
intel	Set of information that virtual agent has
security	The attack strength of the agent.

Table 5: Virtualmap file columns explanation, part 2, configuration

Option	Meaning
capability	Integer. Required for all adversaries. It represents an adversary's capability to compromise grid agents. An adversary can only compromise those grid agents with a lower security than the adversary's capability.
target	Integer, market agent's ID. Required by adversaries: AdvDarth, AdvElvira, and AdvFaust. It indicates the market that will be attacked.
trader	Integer, trader agent's ID. Required by the adversary AdvDarth. It indicates the trader that will be compromised for attack.
shift	Integer. Required by adversaries: AdvDarth, AdvElvira.
reduction	Integer. Range 1-99. Required by the adversary AdvFaust. It indicates the percentage amount by which the bids price will be decreased when the adversary forges fake messages.

4.2.1 Example:

The following example is the configuration of virtual agent “AdvBeth” from file “virtualmapb.csv”:

Table 6: Example configuration of AdvBeth in virtualmapb.csv

Line in File:

```
1001,AdvBeth,capability:0,type1|type2|type3,1|2|3|4|5|201,1|2|3|4|5|201,100
```

Column	**Value**	**Meaning**
id	1001	This virtual agent’s id is 1001.
type	AdvBeth	This virtual agent is adversary “AdvBeth”.
configuration	capability:0	This virtual agent’s capability is 0, which means it is not able to compromise any grid agent.
channel	type1|type2|type3	This virtual agent has access to the type1, type2, and type3 communication channel.
agent	1|2|3|4|5|201	This virtual agent has access to the grid agents 1, 2, 3, 4, 5, and 201.
intel	1|2|3|4|5|201	This virtual agent has information from the grid agents 1, 2, 3, 4, 5, and 201.
security	100	The agent can successfully attack targets with security below 100.

4.2.2 Example

The following example is the configuration of virtual agent “AdvDarth” from file “virtualmapd.csv”.

Table 7: Example configuration of AdvDarth in virtualmapd.csv

Line in File:

```
1001,AdvDarth,capability:100|target:201|trader:95|shift:100,type1|type2,95,95|
201,100
```

Column	**Value**	**Meaning**
id	1001	This virtual agent’s id is 1001.
type	AdvDarth	This virtual agent is adversary “AdvDarth”.
configuration	capability:100| target:202| trader:95| shift:100	This virtual agent’s capability is 100. This virtual agent will attack market agent 202. This virtual agent will try to compromise trader 191. The demand curve will be shifted by 100.
channel	type1|type2	This virtual agent has access to the type1 and type2 communication channels.
agent	95	This virtual agent has access to the grid agent 95.
intel	95|201	This virtual agent has information from the grid agents 95 and 201.
security	100	The agent can successfully attack targets with security below 100.

4.2.3 Example

The following example is the configuration of virtual agent “AdvElvira” from file “virtualmape1.csv”.

Table 8: Configuration of AdvElvira in virtualmape1.csv

Line in File:

```
1001,AdvElvira,capability:100|target:202|shift:100,type1|type2,191,191|
202,100
```

Column	Value	Meaning
id	1001	This virtual agent’s id is 1001.
type	AdvElvira	This virtual agent is adversary “AdvElvira”.
configuration	capability:100| target:202| shift:100	This virtual agent’s capability is 100. This virtual agent will attack market agent 202. The demand curve will be shifted by 100.
channel	type1|type2	This virtual agent has access to the type1 and type2 communication channels.
agent	191	This virtual agent has access to the grid agent 191.
intel	191|202	This virtual agent has information from the grid agents 191 and 202.
security	100	The agent can successfully attack targets with security below 100.

4.2.4 Example

The following example is the configuration of virtual agent "AdvFaust" from file "virtualmapf.csv":

Table 9: Example configuration of AdvFaust in virtualmapf.csv

Line in File:

```
1001,AdvFaust,capability:100|target:202|reduction:30,type1|type2,202,202,100
```

Column	Value	Meaning
id	1001	This virtual agent's id is 1001.
type	AdvFaust	This virtual agent is adversary "AdvFaust".
configuration	capability:100| target:202| reduction:30	This virtual agent's capability is 100. This virtual agent will attack market agent 202. Bid prices will be decreased by 30%.
channel	type1|type2	This virtual agent has access to the type1 and type2 communication channel.
agent	202	This virtual agent has access to the grid agent 202.
intel	202	This virtual agent has information from the agent 202.
security	100	The agent can successfully attack targets with security below 100.

4.3 File testdraw.csv

Table 10: File testdraw.csv columns explanation

Column	Meaning
n	Draw number
type	Type of agent
load	Load at base price
elast	Absolute value of price elasticity

5 Output

5.1 Log Files

A log file records all the messages reported by the simulator during runtime. It contains the following data: (1) the configuration of this scenario; (2) solution steps that have been reached in each population; (3) results after each population has been solved; (4) bids dropped in each population; and (5) additional debugging messages.

5.2 Net Demand Files / Debugging Files

Table 11: Net demand file columns explanation

Column	Meaning
pop	Population
id	Unique identifier of the grid agent.
tag	PW
dos	PW
steps	Demand steps the grid agent has in one dos from each population.
p	Price of the current demand step.
q_min	Minimum quantity of the current demand step.
q_max	Maximum quantity of the current demand step.

5.3 Output Files

"Output" file records grid agent's price, actual quantity, upstreaming trans- mission constraint, and some other runtime information after each simulation.

Table 12: Output file columns explanation

Column	Meaning
pop	Population
dos	DOS level
id	Unique identifier of the grid agent.
rblock	Random variable used for determining blocking.
blocked	Whether the agent's bids were blocked.
p	Grid agent's price record after one simulation.
q	Grid agent's actual quantity record after one simulation.
upcon	If the grid agent's upstream transmission is binding or not. N = No constraint; S = At maximum supply; D = At maximum demand.

5.3.1 Example

The following example is an output record from the output file "c2k1800 out.csv".

Table 13: Example record from output file ck21800 out.csv

Line in File:

```
1,0,1,41.3,0,70,40,N
```

Column	Value	Meaning
pop	1	This record comes from population 1.
dos	0	This run had DOS 0.
id	1	This record belongs to the grid agent 1.

rblock	41.3	Random variable used for determining blocking
blocked	0	Agent was not blocked.
p	70	The price of grid agent 1 is 70.
q	40	The actual quantity of grid agent 1 is 40.
upcon	N	No constraint exists in the upstream transmission.

6 Modeling Framework

6.1 Simulation Environment

The simulator first loads run configurations into the simulation environment. "Env" object will check settings from the configuration file, then instantiate agents in the market model using corresponding data files.

Environment is also responsible for storing critical variables and information for the simulation. Some of them are defined as "static" in order that these data can be shared with other objects. Stage change is also controlled by the environment. It starts up the simulation from the stage SERVICE SEND, then to the stage TRADER SEND, PRE AGGREGATE, AGGREGATE, PRE REPORT, REPORT, PRE CALC LOADS and the simulation eventually terminates at the stage CALC LOADS. Different kinds of agents work in different stages:

Trader agents send demand messages to the upstream market during the stage TRADER SEND, and receive price messages and determine the loads during the stage CALC LOADS.

Market agents aggregate demand messages from the downstream agents and send it to the upstream market at the stage AGGREGATE. Then they will receive price messages from the upstream market and report the price to downstream agents during the stage REPORT. And they also calculate the actual loads during the stage CALC LOADS.

Adversaries can hook into the "PRE" stages to intercept the channels and send fake messages. These "PRE" stages happen before the corresponding stages and are only used to simulate attack behaviors.

More details can be found in the Object Reference Section "Env".

6.2 Traders and Markets

Traders and markets are grid agent nodes connected by both physical connections and virtual connections. Trader agent can work as a demander to consume electricity or as a supplier to provide electricity. Traders interact with their upstream market using communication channels.

Market agent is responsible for demand aggregation and electricity rates determination. There are several tiers of market agents in the model:

Root market agent calculates the electricity price based on the demand messages from the downstream child agents.

Intermediate market agent utilizes the price message from its up- stream market and other transmission parameters, such as transmission cost and capacity limit, to determine the actual price, and sends the actual price to downstream child agents.

More details can be found in the Object Reference Sections “Trader“, “Market“, and “Demand“.

6.3 Communications

Each grid agent in the market model does not only has physical connection to its upstream market agent but also has a virtual connection, which is called communication channel, with the upstream object. Note that this is not applied to the market agent on the top level since it does not have an upstream market agent.

The physical connection is used to simulate transmission lines and the communication channel is for transmitting messages between grid agents and the upstream markets. Three types of communication channels are being used in the simulation: type1, type2, and type3 channel. Virtual agents are able to connect with the market and send messages through communication channels as well.

There are three kinds of messages transmitted through channels: empty, demand, and price message. Demand messages are sent by a grid agent to its upstream market. Price messages come from upstream market to downstream agents.

More details can be found in the Object Reference Sections “Channel“ and “Msg“.

6.4 Simulating a Denial of Service (DOS) Attack

We simulate this process by randomly dropping bids during message transmission through channels. The percentage of the dropped bids is given by the configuration file.

6.5 Adversaries

A virtual agent is a special market node that does not have physical connections to other grid agents but is able to build virtual connections with grid agents through communication channels. The Adversary class is a kind of virtual agent that can access sensitive information and perform attacks based on these information. They can access historical bids, demands data, and other intel from some grid agents, and even compromise the grid agent. They can also interfere the system by intercepting the channels and steal runtime information, such as message being transmitted in the channel. Then they utilize these information to forge messages and spread the fake messages in the market to destruct the electricity and price control.

More details can be found in the Object Reference sections “Adversary“, “History“, and “Intel“.

7 Objects Reference

7.1 Adversary

Adversary is an abstract class for virtual agents who try to disrupt the power grid. It has several subclasses that can perform different kinds of attacks.

7.1.1 AdvAdam

An AdvAdam object can send false bids to the other nodes inside the grid. It looks up the Intel hash map and picks all the target nodes that be connected to. Then constructs the false bid and sends the bid to the targets.

7.1.2 AdvBeth

AdvBeth class is the promoted version for AdvAdam class. It has all the functionalities provided by AdvAdam as well as to forge credentials and to deceive the recipient with that.

7.1.3 AdvDarth

An AdvDarth object is able to attack the market with constrained transmission to their upstream nodes. It is supposed to have compromised one trader from the target market. It first intercepts all messages sent by the compromised trader. Then extracts the trader's historical demands. Shifts the demand curve by the attacker customized distance. Finally injects the fake demands back into the channel and these demands will be sent to the target market.

7.1.4 AdvElvira

AdvElvira performs similar behavior to AdvDarth. We suppose that AdvElvira has compromised several traders instead of only one. Then it performs the attack from all of these compromised traders, and shifts all their demand curves by a smaller value, with a total shift distance that equals to the attacker customized distance. Compared with the Darth, Elvira performs the attack more softly but with more traders to achieve the same result, which means it has less chance of being discovered by an anomalous data detection algorithm.

7.1.5 AdvFaust

An AdvFaust object listens the target market channel and intercepts messages sent by the target. Then injects the DEMAND message back to the channel without any modification, but tampers the price info in the PRICE message and injects it back to the channel before traders calculate loads.

7.2 Agent

Agent is an abstract class that provides basic features for entities that communicate. It has two subclasses: Grid, for agents actually connected to the power grid, and Virtual, for agents that only have communication links.

7.3 Channel

A Channel object is communications channel. An arbitrary number are allowed and they can have different properties. Each agent has a specific Channel that it uses to communicate with its upstream parent node. A Channel is used to by one agent to send Msg objects to another. Each Channel has one main method, send(). It looks up the sender and recipient from the corresponding fields of the Msg object it is given and checks whether random denial of service filtering applies to the sender. If so, the message is dropped. Otherwise, as long as the message is not diverted (discussed below) it is passed to the recipient via the recipient's deliver() method. Three hooks are available to support man in the middle attacks and other interventions. Message diversions can be set up via each channel's divert_to() and divert_from() methods. The first diverts all messages sent to a given node and the second diverts all messages sent *by* a given node. The from diversion is processed first and when both apply to a given message it takes precedence. Messages can be reinserted downstream from the diversions via the channel's inject() method. Future features to be implemented:

- Random DOS loss rates that can vary by channel
- VPN channels that prohibit diversions
- Authentication of senders
- Authentication of recipients

7.4 Demand

A Demand object holds a net demand curve expressed as a list of steps. Positive quantities indicate demand and negative quantities indicate supply. Trader nodes send Demand objects to Market nodes. Lower tier Market nodes send aggregated Demand objects to higher-tier Market nodes. In all cases the curves are sent via Msg objects.

7.5 Env

The Env object represents the environment under which the simulation is running. It includes various global variables and is also responsible for loading data, configuring the network of Agent nodes, the Channel objects they use to communicate, and then starting the simulation.

7.6 Grid

Abstract class for grid-connected agents (that is, agents through which power can flow). Has two subclasses: Trader and Market.

7.7 History

A History object stores historic price and quantity data for a specific agent. Provides several methods used to store and retrieve demand, price, quantity, and constraint information. The simulator can load historic data from preset history file before it runs. Adversaries can extract relevant in- formation from history object to build fake demand curves and then perform attacks.

7.8 Intel

Stores a virtual agent's information about another agent within the grid. Contains functionality to store an agent's grid-level (transmission cost, price, parent, children, tier), historic (price, quantity, bid), and some status (compromised, interceptTo, interceptFrom, forge, send ,learned) information. Can also retrieve max/min/avg information regarding p and q.

7.9 Market

Represents a market. Aggregates demands by child nodes, which can be Trader or other Market nodes. If the node has a parent, it adjusts the aggregate demand for transmission costs and capacity to the parent and passes the adjusted demand up. If the node does not have a parent, it determines the equilibrium price. Market nodes receive prices from upstream, adjust them for transmission parameters, and then pass them down to child nodes.

7.10 Msg

A single message from one agent to another. At the moment, two types of messages can be sent: one with a Demand object and one with a price. All Msg objects are passed via Channel objects.

7.11 Trader

Abstract class for end agents. Trader nodes have upstream parents that are Market nodes. They submit Demand objects to their parent Market nodes and then receive prices back. Final demand or supply results from the prices received. Provides method getOneDemand() to retrieve demand, static method readDraws() to load draws from history files, and abstract method drawLoad() to build the agent's demand curve.

Has subclass: TraderMonte.

7.11.1 TraderMonte

A TraderMonte object represents the trader under Monte Carlo mode. Has two types: end users and suppliers. Implements method drawLoad(), and provides method readDraws().

7.12 Util

A utility class that includes a few general purpose methods for opening files with built-in exception handling.

7.13 Virtual

Abstract class for agents connected to the communications network but not connected directly to the power grid.