SaTC Reference Manual

A

March 5, 2019

Contents

1 Overview

2 Installation

SaTC project depends on two external libraries:

- **commons-csv-1.4.jar**
  Commons CSV reads and writes files in variations of the Comma Separated Value (CSV) format. More details can be found on the project official website: Commons CSV Home

- **mason.19.jar**
  SaTC project 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: Mason

To build the SaTC project:

- 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', you need 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:

     $$\text{javac } -\text{cp } '../\text{lib}/*;.' \ \$*.\text{java}$$

  3. Generate the java archive file with the following command:

     $$\text{jar cvfm market.jar manifest.txt } *.\text{class}$$

  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

You should build the SaTC project firstly before running the simulator. Build reference can be found in the section 'Installation'.

If you have built the SaTC project 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 the simulation:

1. Launch the command line and change the working directory to the directory 'run'.
2. Execute the following command:

   $$\text{java } -\text{jar } ../\text{src}/\text{market.jar } < \text{run\_configuration\_file} >$$

   More details about the run configuration files can be found in the section: Run Configuration Files

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 file options

option	**meaning**	**type**
netmap	Path of the grid agent configuration file.	Required
virtualmap	Path of the virtual agent configuration file.	Optional
draws	Path of the draw file.	Required
history	Path of the historical output file.	Optional
bids	Path of the bid file.	Optional
transcost	Transmission cost between nodes.	Optional
transcap	Maximum transmission between nodes.	Optional
seed	Seed for random number generator.	Optional
debug	Debugging flag (0, 1, default: 0).	Optional

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

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.

Example The following example is from the configuration file 'c2k1800d.txt'. This file is used for simulation with adversary 'Adv_Darth'. 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 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: Traders.)
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.

Example The following example is the configuration of node 1:

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

Table 3: Configuration of node 1 in netmap.csv

column	**value**	**meaing**
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 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

column	**meaning**
id	Unique identifier of virtual agents.
type	Virtual agent type, including Adv_Adam, Adv_Beth, Adv_Darth, Adv_Elvira, Adv_Faust.
configuration	Configuration parameters for virtual agent's behavior. Please see below.
channel	Communication channels the virtual agent has access to.
agent	ID of grid agents that virtual agent has access to.
intel_level	How much information the virtual agent has. 'full' means the virtual agent has intel from all the grid agents. 'partial' means the virtual agent has intel from the grid agents in its intel set.
intel	Set of information that virtual agent has
security	KC

The table below introduces configuration parameters used by virtual agents.

Table 5: Configuration options

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: Adv_Darth, Adv_Elvira, and Adv_Faust. It indicates the market that will be attacked.
trader	Integer, trader agent's ID. Required by the adversary Adv_Darth. It indicates the trader that will be compromised for attack.
shift	Integer. Required by adversaries: Adv_Darth, Adv_Elvira. PW
reduction	Integer. Range 1-99. Required by the adversary Adv_Faust. It indicates the percentage amount by which the bids price will be decreased when the adversary forges fake messages.

Example The following example is the configuration of virtual agent 'Adv_Beth' from file 'virtualmapb.csv'.

```
1001,Adv_Beth,capability:0,"type1,type2,type3","1,2,3,4,5,201",
    partial,"1,2,3,4,5,201",100
```

Table 6: Configuration of Adv_Beth in virtualmapb.csv

column	**value**	**meaning**
id	1001	This virtual agent's id is 1001.
type	Adv_Beth	This virtual agent is adversary 'Adv_Beth'.
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_level	partial	This virtual agent is able to obtain intel from the grid agents inside its intel set.
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	KC

Example The following example is the configuration of virtual agent 'Adv_Darth' from file 'virtualmapd.csv'.

```
1001,Adv_Darth,"capability:100,target:202,trader:191,shift
    :100","type1,type2","191",full,"191,202,203",100,none
```

Table 7: Configuration of Adv_Darth in virtualmapd.csv

column	value	meaning
id	1001	This virtual agent's id is 1001.
type	Adv_Darth	This virtual agent is adversary 'Adv_Darth'.
configuration	capability:100, target:202, trader:191, 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 channel.
agent	191	This virtual agent has access to the grid agent 191.
intel_level	full	This virtual agent is able to obtain intel from all the grid agents.
intel	191,202,203	This virtual agent has information from the grid agents 191, 202, and 203.
security	100	KC

Example The following example is the configuration of virtual agent 'Adv_Elvira' from file 'virtualmape1.csv'.

```
1001,Adv_Elvira,"capability:100,target:202,shift:100","type1,
    type2",191,full,"191,202",100,none
```

Table 8: Configuration of Adv_Elvira in virtualmape1.csv

column	**value**	**meaning**
id	1001	This virtual agent's id is 1001.
type	Adv_Elvira	This virtual agent is adversary 'Adv_Elvira'.
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 channel.
agent	191	This virtual agent has access to the grid agent 191.
intel_level	full	This virtual agent is able to obtain intel from all the grid agents.
intel	191,202	This virtual agent has information from the grid agents 191 and 202.
security	100	KC

Example The following example is the configuration of virtual agent 'Adv_Faust' from file 'virtualmapf.csv'.

```
1001,Adv_Faust,"capability:100,target:202,reduction:30","type1,
    type2","202",full,"202",100,none
```

Table 9: Configuration of Adv_Faust in virtualmapf.csv

column	**value**	**meaning**
id	1001	This virtual agent's id is 1001.
type	Adv_Faust	This virtual agent is adversary 'Adv_Faust'.
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_level	full	This virtual agent is able to obtain intel from all the grid agents.
intel	202	This virtual agent has information from the agent 202.
security	100	KC

4.3 testdraw.csv

Table 10: Draw file columns explanation

column	**meaning**
n	PW
type	PW
load	PW
elast	PW

5 Output

5.1 Log Files

A log file records all the messages reported by the simulator during runtime.
It contains:

- Configuration of this scenario.
- Steps that have been reached in each population.
- Results after each population.
- Bids dropped in each population.
- Some other debugging messages.

5.2 Net Demand Files / Debugging Files

Table 11: Net demand file columns explanation

column	**meaning**
pop	PW
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 transmission constraint, and some other runtime information after each simulation.

Table 12: Output file columns explanation

column	**meaning**
pop	PW
dos	PW
id	Unique identifier of the grid agent.
rblock	PW
blocked	PW
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.)

Example The following example is an output record from the output file 'c2k1800_out.csv'.

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

Table 13: A record from output file ck21800_out.csv

column	**value**	**meaning**
pop	1	This record comes from population 1.
dos	0	This record got after dos 0.
id	1	This record belongs to the grid agent 1.
rblock	41.3	PW
blocked	0	PW
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

Simulator firstly loads run configurations into the simulation environment. 'Env' object will check settings from the configuration file, then initiate 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 section 'Env'.
// 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 upstream 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 section 'Trader', 'Market', and 'Demand'. // Trader, market, demand, net demands

6.3 Price Determination

Transmission costs, congestion, aggregation, population

6.4 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 section ‘Channel’ and ‘Msg’.
// Channel, msg

6.5 DoS

DoS is the abbreviation standing for the term, ‘Denial of Service’. 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. Note that this attack is more severe in the real world with a higher bids dropped rate.

6.6 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.

Adversary is a kind of virtual agent that can access senstive 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 section ‘Adversary’, ‘History’, and ‘Intel’.
// History, 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.

Adv_Adam

An **Adv_Adam** 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.

Adv_Beth

Adv_Beth class is the promoted version for **Adv_Adam** class. It has all the functionalities provided by **Adv_Adam** as well as to forge credentials and to decipt the recipient with that.

Adv_Darth

An **Adv_Darth** 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.

Adv_Elvira

Adv_Elvira performs similar behavior to **Adv_Darth**. We suppose that **Adv_Elvira** 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 possbility to be discovered by the anomalous data detection.

Adv_Faust

An **Adv_Faust** 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 – blocking **divert_to()**

7.4 Demand

A **Demand** object holds a net demand curve expressed as a list of steps. Positive quantities indicate demand and negative quantites 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 information 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. Future features to be implemented:

- Encryption that prevents reading by diverters
- Signing that prevents spoofing the sender

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**.

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.