

# CS682: IOT-Simulator-Platform User Document

Author: Bang Tran, Chenjun Li, Yiwei Yao

Github: <https://github.com/chenjunlee/IoT-Simulation-Platform>

This Document should be worked with CupCarbon User Guide together.

[http://labsticc.univ-brest.fr/~bounceur/cupcarbon/doc/cupcarbon\\_user\\_guide.pdf](http://labsticc.univ-brest.fr/~bounceur/cupcarbon/doc/cupcarbon_user_guide.pdf)

# Table of Contents

## 1. Users Implement

- 1.1. Name
- 1.2. Selected Area
- 1.3. Area of Interests
- 1.4. Quality of Services
- 1.5. Running Period
- 1.6. Secure Mode

## 2. NetWork

- 2.1. Randomly generate Network
- 2.2. Routers
- 2.3. Sensors
- 2.4. Nature Events
- 2.5. Generate Network Using Python Script for customization

## 3. DataBase

- 3.1. Import to DataBase
- 3.2. Export from DataBase

## 4. Result

- 4.1. GUI Show Result
- 4.2. PRINTDB

## 5. New Script Command

- 5.1. AREADSENSOR
- 5.2. LED
- 5.3. GETU
- 5.4. PRINTDB

# 1. Users Implement

User is a spot that has its location and can be placed everywhere on the map.

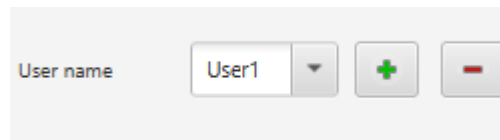
Once User is created, it will connect with its nearest Base Station, and get/send data from/to this Base Station.

User Object (user/User.java) has properties:

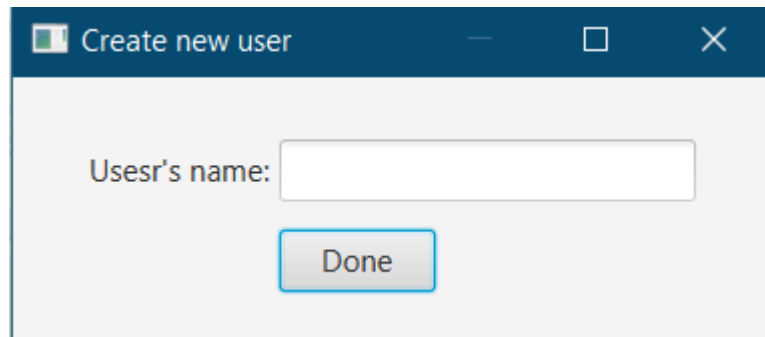
- Name
- Selected Area
- Area of Interests
- Quality of Services
- Running Period
- Secure Mode

## 1.1. Name:

In order to create a user, you can click the plus button, on the other hand, if you want to delete a user click minus button. The selected user can be switched using down arrow:

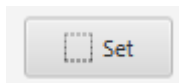


each user has its identified name from each other, and it can be set when creating a user object:

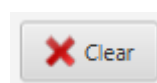


## 1.2. Selected Area:

Selected Area can be set by putting two marks  that decides the area, and click

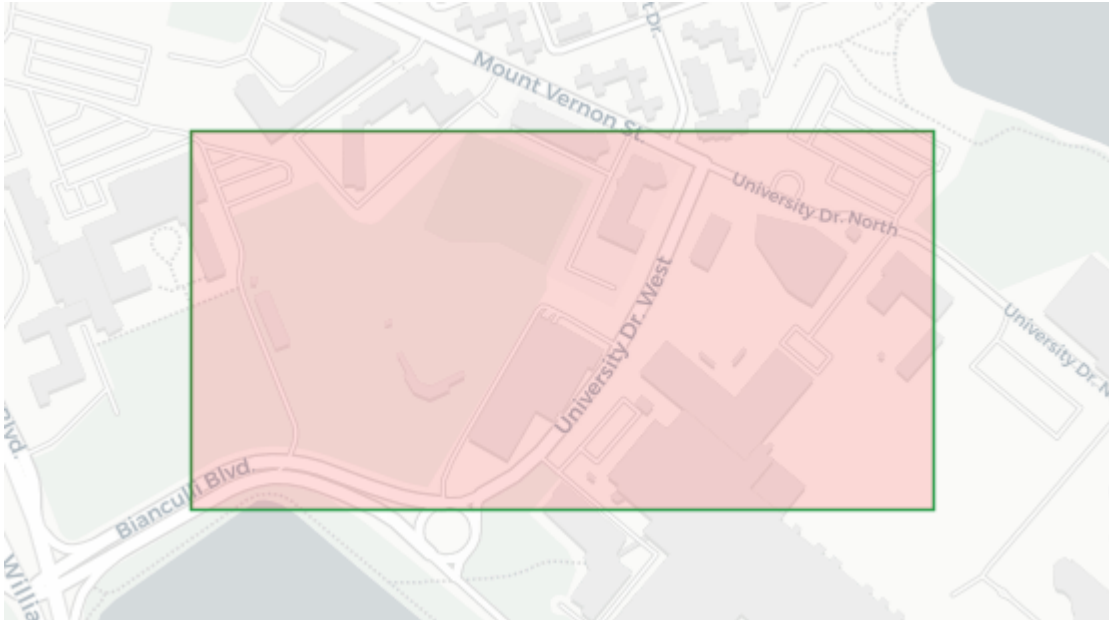


button, and clear the area by clicking

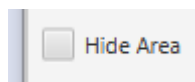
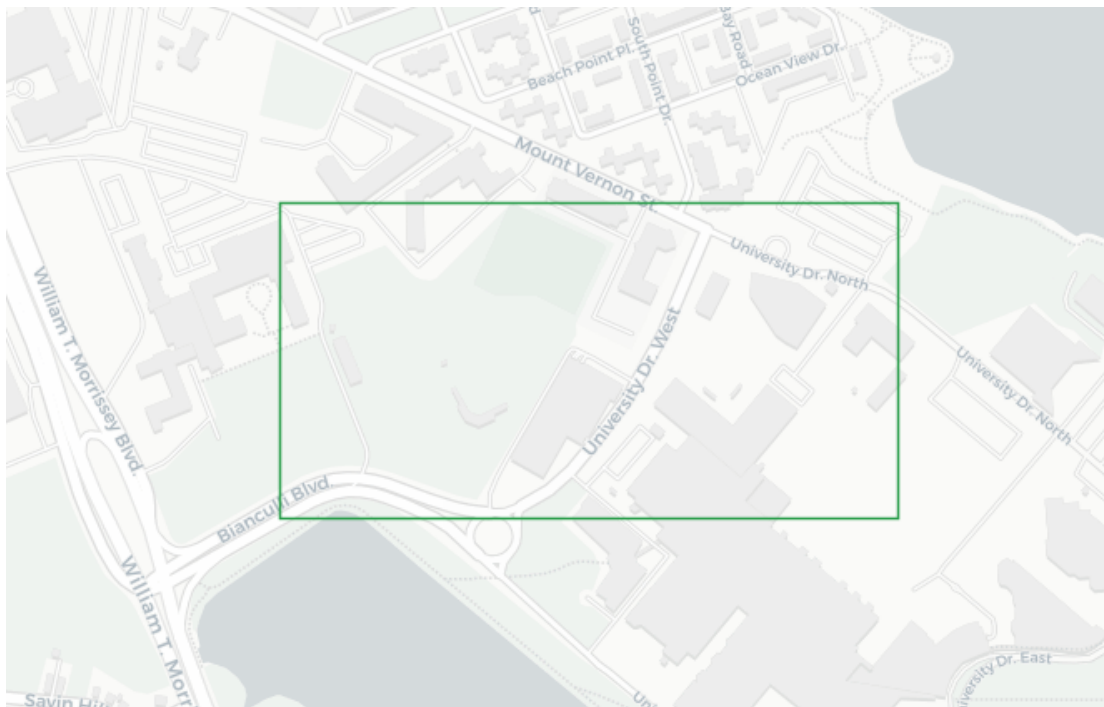


button.

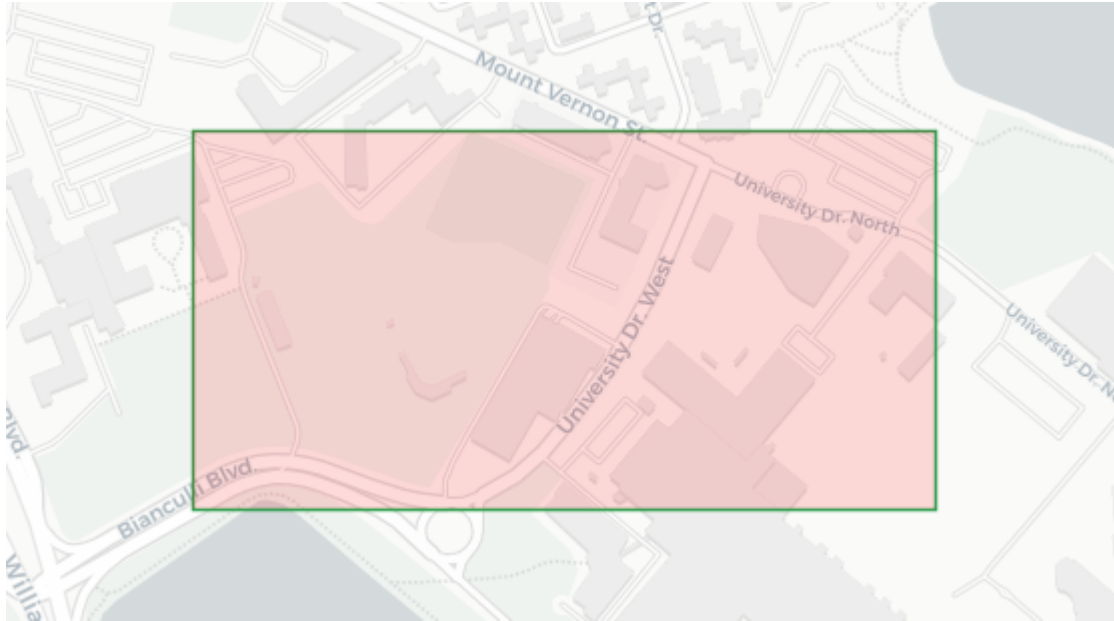
Selected Area is the area that this user is interested in and wants to get data from. If user1 is selected then the Selected Area will show up like this:



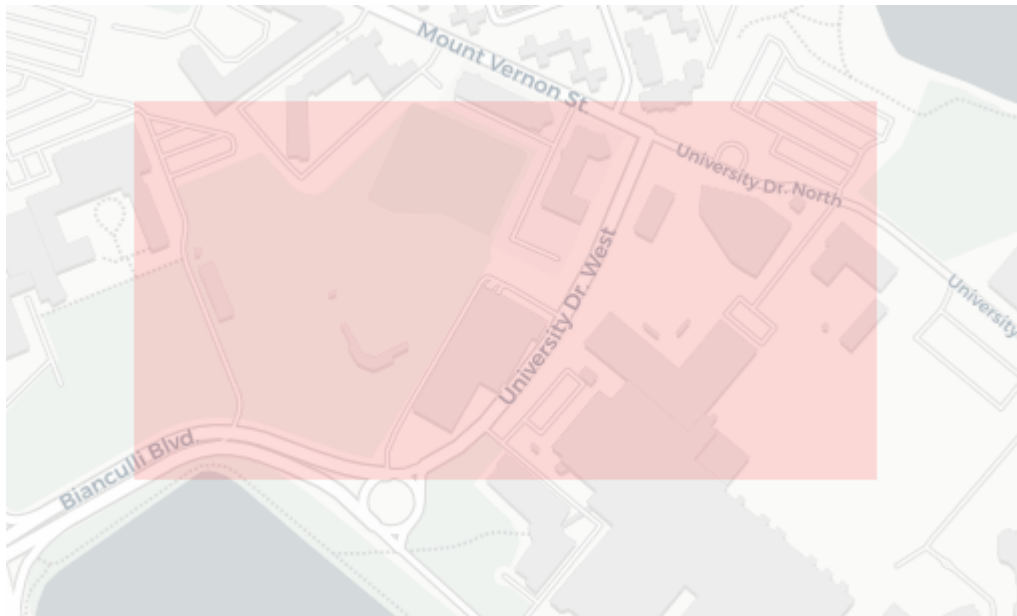
Otherwise, it will show like this:



What's more, you can select ☐ Hide Area to remove the board of selected Area:  
Before:



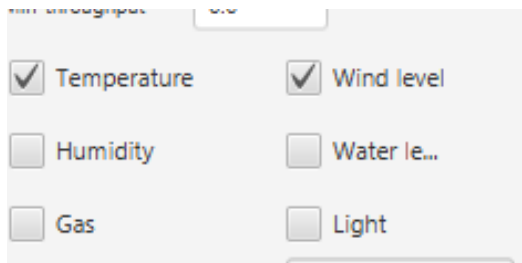
After:



### 1.3. Area of Interests:

Area of Interests gives users the options of what kind of data they are interested in, we provide six types of data. They are Temperature, Wind Level, Humidity, Water Level, Gas, Light. The user can choose the data he is interested in by checking the check-box or deletes his interest by unchecking it.

Area of interests check box:

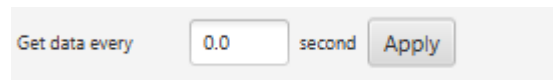


A screenshot of a web interface showing a list of environmental parameters with checkboxes. The parameters are arranged in two columns. The first column contains 'Temperature', 'Humidity', and 'Gas'. The second column contains 'Wind level', 'Water le...', and 'Light'. The 'Temperature' and 'Wind level' checkboxes are checked, while the others are unchecked.

<input checked="" type="checkbox"/> Temperature	<input checked="" type="checkbox"/> Wind level
<input type="checkbox"/> Humidity	<input type="checkbox"/> Water le...
<input type="checkbox"/> Gas	<input type="checkbox"/> Light

#### 1.4. Quality of Service:

Quality of Service gives the option for retrieve data from Base Station in desired seconds:



A screenshot of a web interface for Quality of Service. It features a label 'Get data every' followed by a text input field containing '0.0', the word 'second', and an 'Apply' button.

Get data every  second Apply

It also gives Maximum latency and min-throughput for users to optimize:



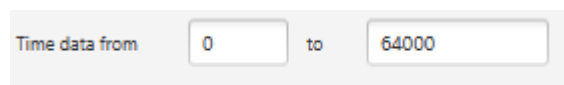
A screenshot of a web interface showing two input fields. The first is labeled 'Maximum latency' and contains the value '10.0'. The second is labeled 'Min-throughput' and contains the value '0.0'.

Maximum latency

Min-throughput

#### 1.5. Running Period:

Running Period allows users to collect data from a certain running time to a certain running time:

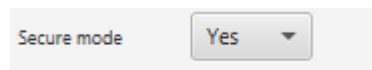


A screenshot of a web interface for Running Period. It shows a label 'Time data from' followed by a text input field containing '0', the word 'to', and another text input field containing '64000'.

Time data from  to

#### 1.6. Secure Mode(Not implemented yet):

Secure Mode allows the user to send encrypt requests to the base station and retrieve encrypt data from the base station.



A screenshot of a web interface for Secure Mode. It shows a label 'Secure mode' followed by a dropdown menu currently displaying 'Yes'.

Secure mode

## 2. NetWork

### 2.1. Randomly generate Network:

We Implement generating networks randomly in two ways, one is a uniform distribution generator, the other is a traditional generator.

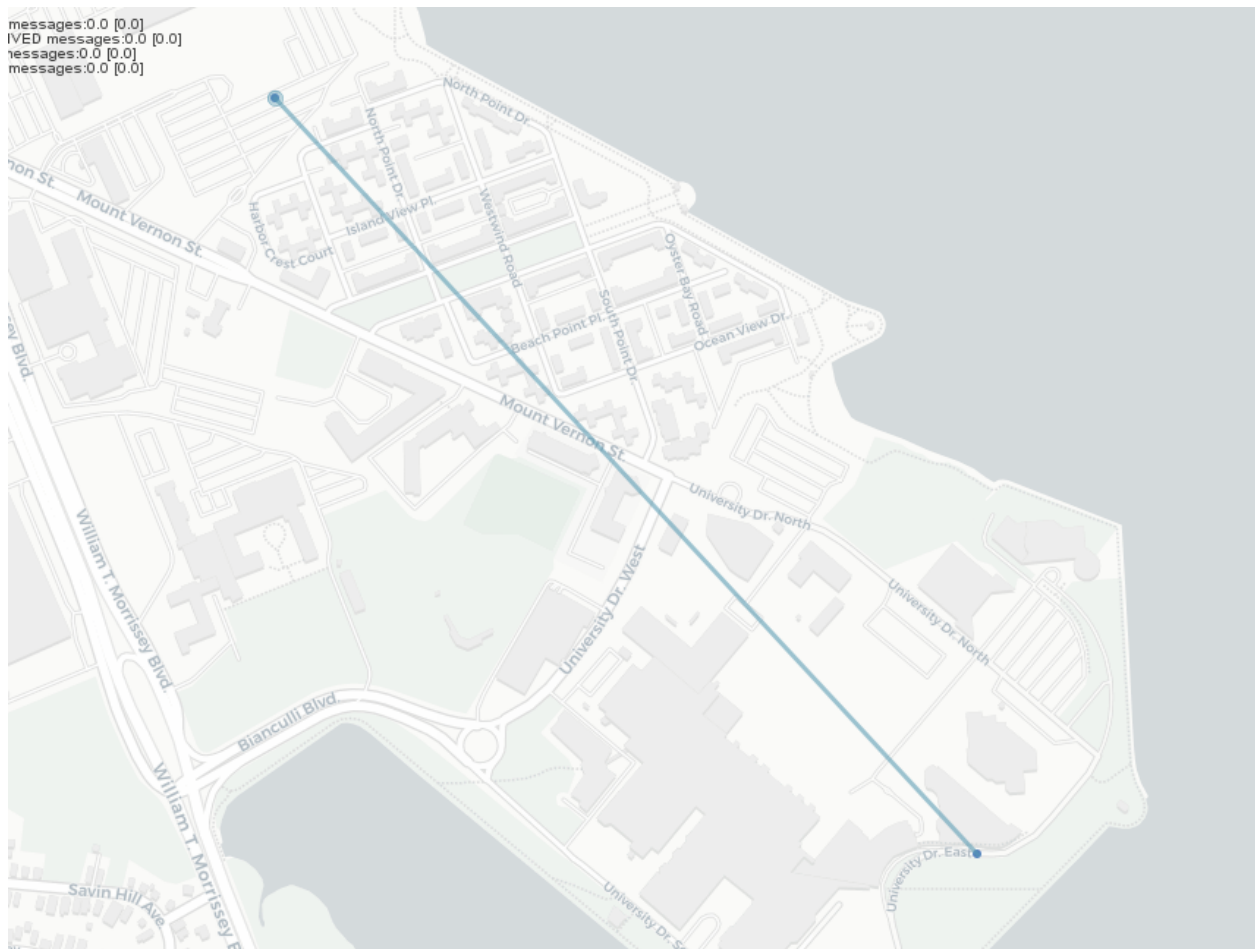
#### Traditional generator:

Check the CupCarbon User Guide.

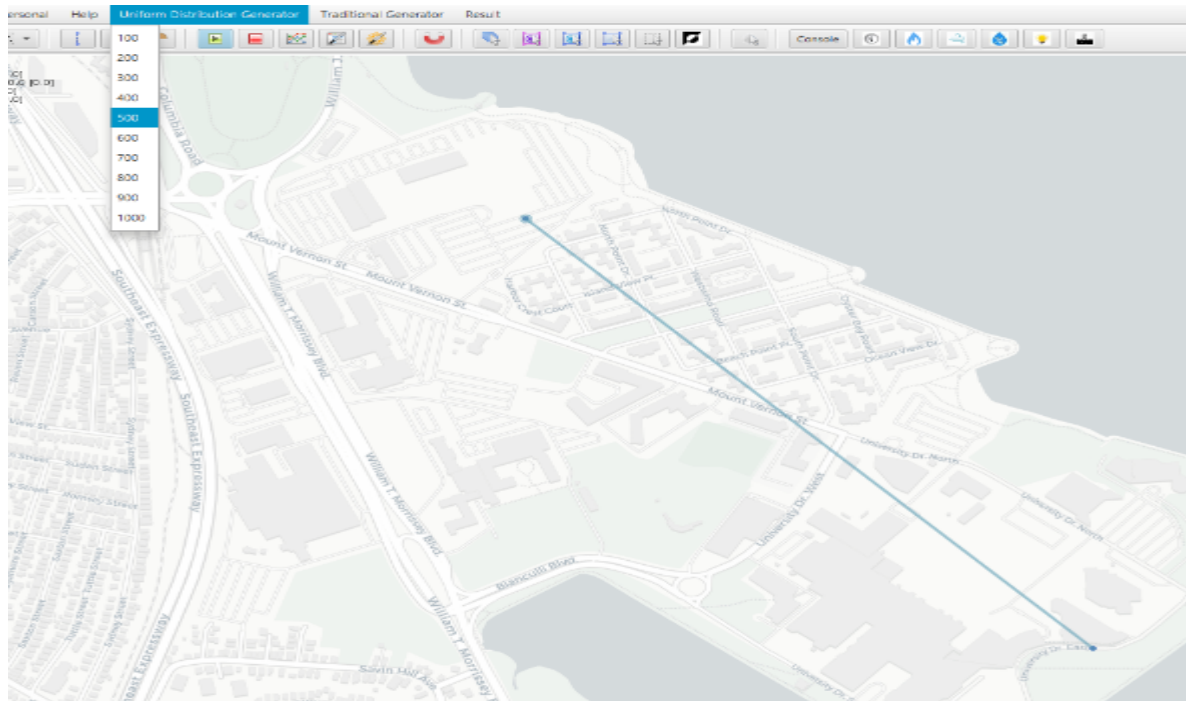
#### Uniform distribution generator:

Uniform distribution generator is based on uniform distribution. In our implementation, we divide a selected rectangle area to 4 parts. Each part has same number of sensors, routers, base stations and nature events. The number of routers will be half of sensors. The number of base stations will be half of routers. The number of nature events will be same as sensors for a better simulation result.

Using mark to choose two points:



Select Uniform distribution generator:



Get the network:





## 2.2. Routers:

Check CupCarbon User Guide.

## 2.3. Sensors:

Check CupCarbon User Guide.

## 2.4. Nature Events:

The old CupCarbon only includes one nature event which is gas. We offer another 5 new nature events -- Temperature, Lighting, Humidity, Water Level, Wind Level.

User can add new nature events by selecting buttons at right up corner on the top toolbar.



## 2.5. Generate Network Using Python Script for Customization:

We also offer an optional way to generate uniform distribution network from python. This feature let user has more freedom to customize his network. There are two python files under pythonGenerateNetworkForMongoDB directory -- generator.py and importToMongoDB.py. User can easily change parameters of function in generator.py to generate network. The function is on the last line of the file.

```
215  
216  
217 genSimulationUnit(-71.0446572303772, 42.319177010972695, -71.03641748428345, 42.31097379616, 100, 2)
```

The first two parameters are the longitude and latitude of point 1. The 3rd and 4th parameters are the longitude and latitude of point 2. The 5th parameter is the number of sensors. The last parameter is the number of users. This way will generate both network and random users.

After running generator.py, the user can run importToMongoDB.py without changing anything. Make sure to check the database if any data need to be saved. This program will clear all the old data. When the generated network is in the database, we can load it directly.

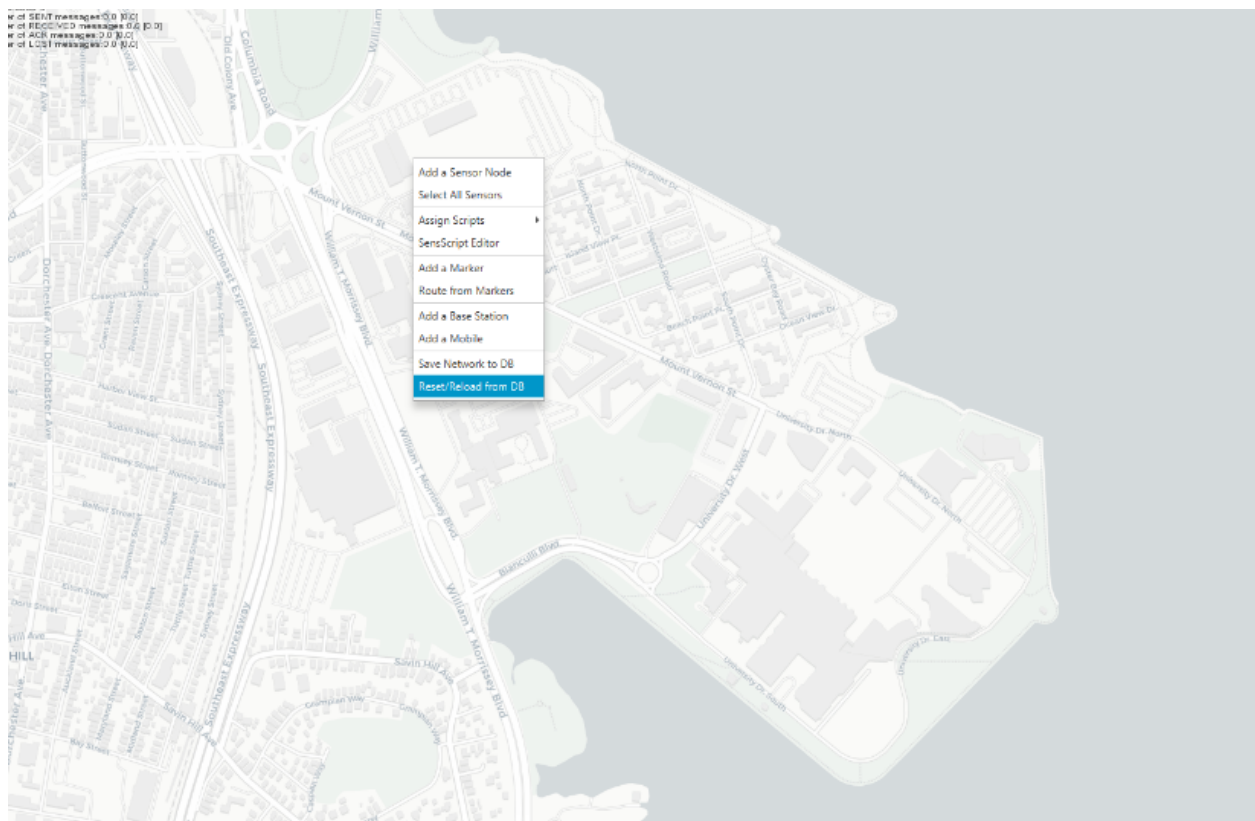
Running python file directly:

```
In [1]: runfile('C:/Users/Chenjun Li/Desktop/test333/IoT-Simulation-Platform/pythonGenerateNetworkForMongoDB/generator.py', wdir='C:/Users/Chenjun Li/Desktop/test333/IoT-Simulation-Platform/pythonGenerateNetworkForMongoDB')

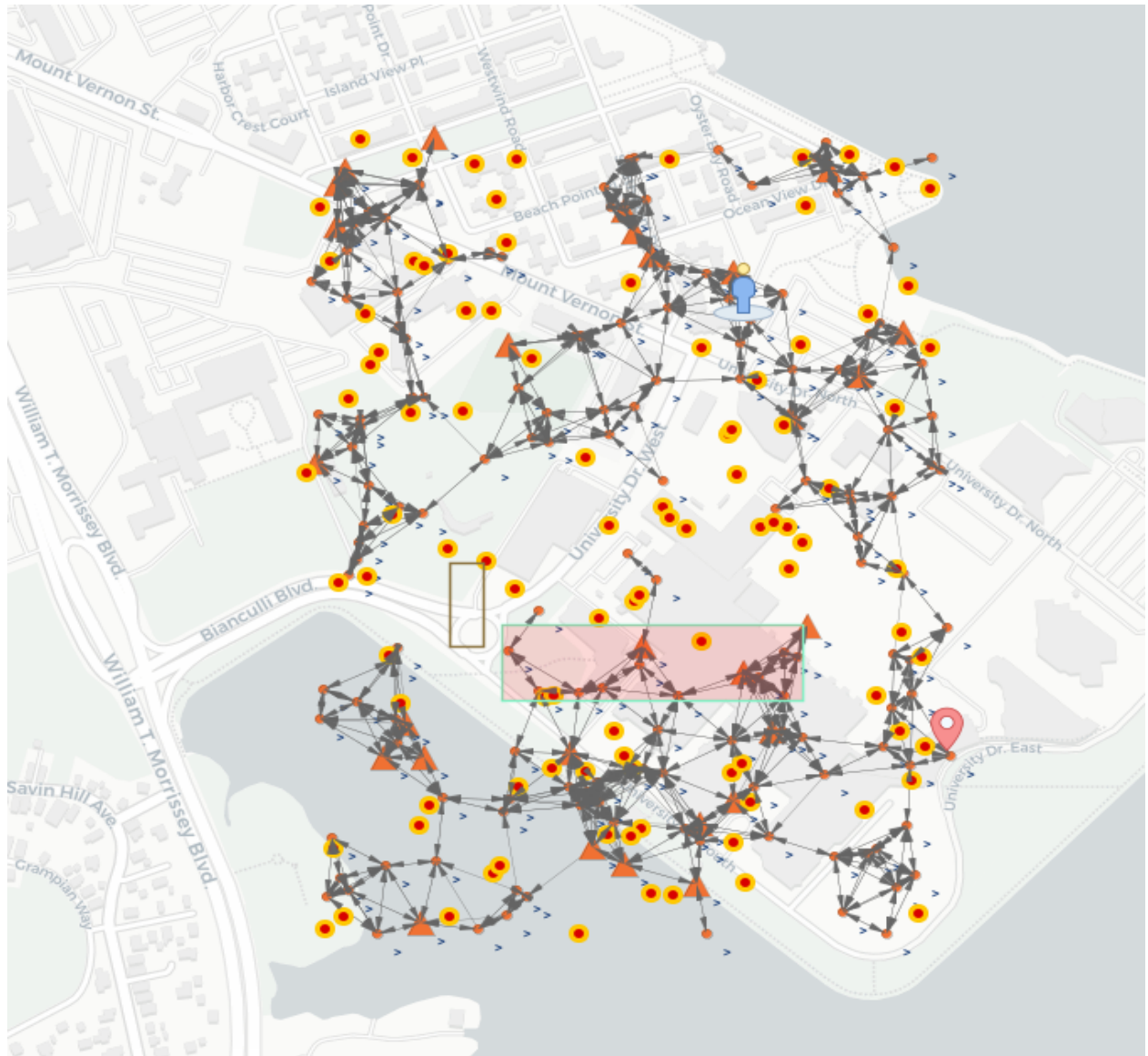
In [2]: runfile('C:/Users/Chenjun Li/Desktop/test333/IoT-Simulation-Platform/pythonGenerateNetworkForMongoDB/importToMongoDB.py', wdir='C:/Users/Chenjun Li/Desktop/test333/IoT-Simulation-Platform/pythonGenerateNetworkForMongoDB')
cs682 exist
460 documents deleted in devices.
2 documents deleted in users.
1 documents deleted in proj_preferences.

In [3]:
```

Right-click mouse and choose Reset/Reload from DB:



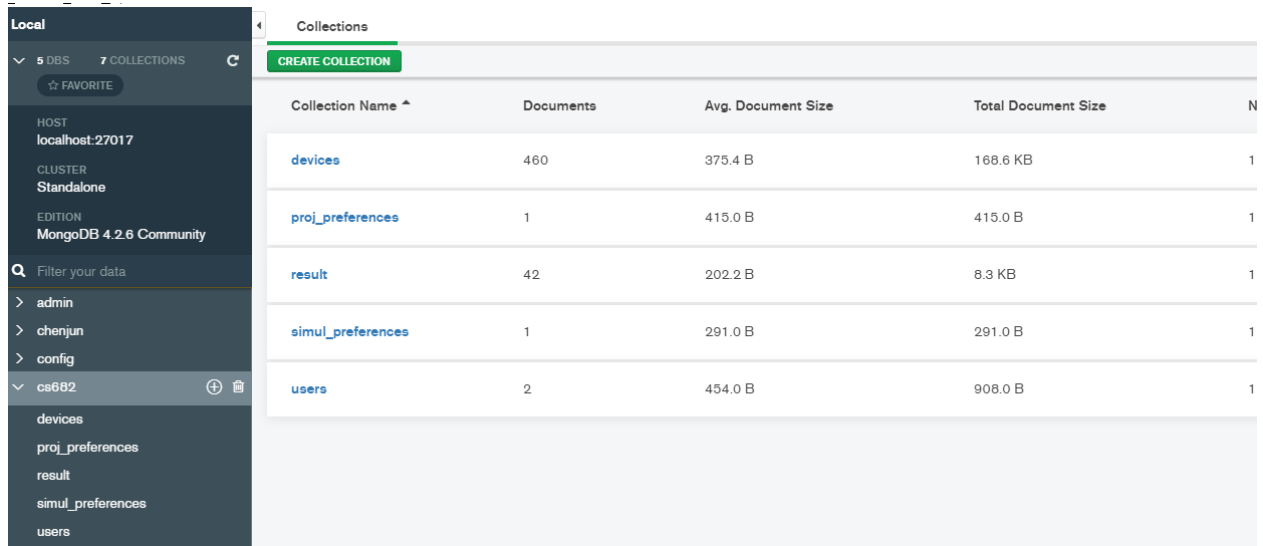
Load network generated by python from database:



### 3. DataBase

We use MongoDB to store our data including network data, user data, and simulation result data.

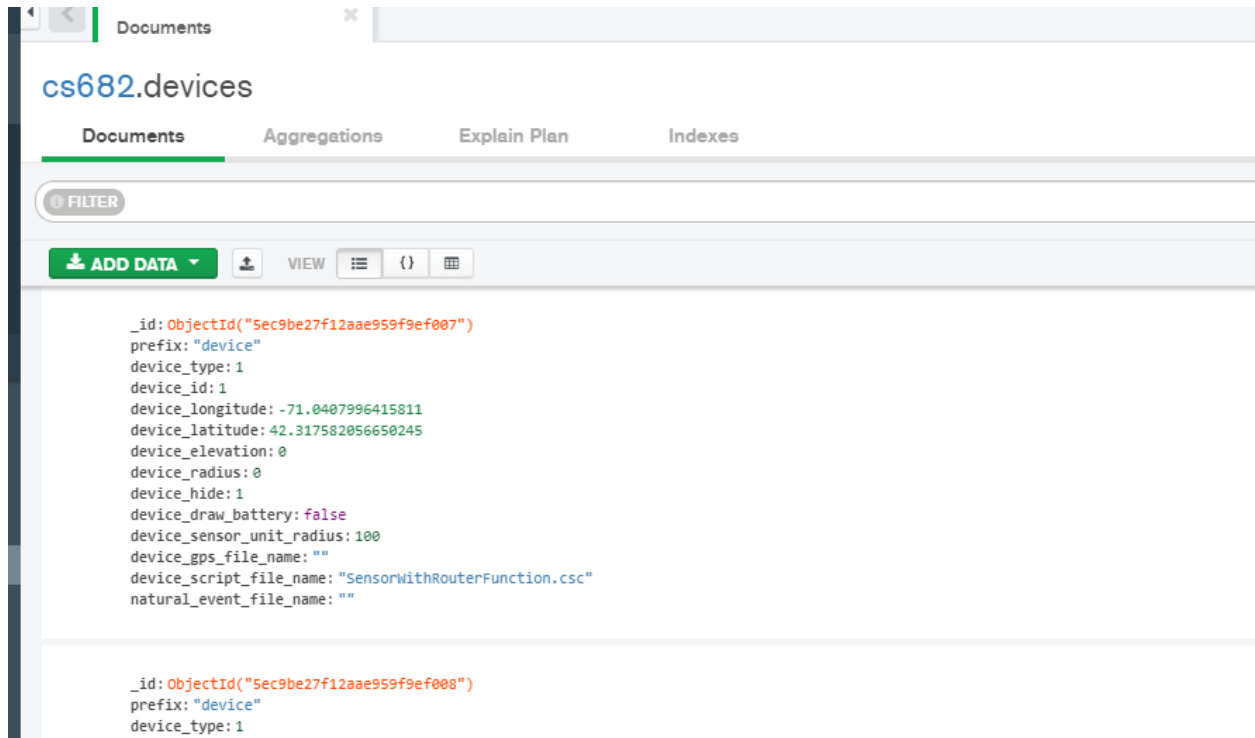
Database structure (cs682 is the database for this project):



The screenshot shows the MongoDB Compass interface. On the left, the 'Local' sidebar displays the database 'cs682' with a list of collections: 'devices', 'proj\_preferences', 'result', 'simul\_preferences', and 'users'. The main panel shows the 'Collections' tab with a table listing these collections and their document counts and sizes.

Collection Name	Documents	Avg. Document Size	Total Document Size	N
devices	460	375.4 B	168.6 KB	1
proj_preferences	1	415.0 B	415.0 B	1
result	42	202.2 B	8.3 KB	1
simul_preferences	1	291.0 B	291.0 B	1
users	2	454.0 B	908.0 B	1

Network data:



The screenshot shows the 'Documents' tab for the 'cs682.devices' collection. It displays two JSON documents representing network data. The first document contains detailed information about a device, including its ID, type, location, and configuration. The second document is partially visible and shows the start of another device record.

```
_id: ObjectId("5ec9be27f12aae959f9ef007")
prefix: "device"
device_type: 1
device_id: 1
device_longitude: -71.0407996415811
device_latitude: 42.317582056650245
device_elevation: 0
device_radius: 0
device_hide: 1
device_draw_battery: false
device_sensor_unit_radius: 100
device_gps_file_name: ""
device_script_file_name: "SensorWithRouterFunction.csc"
natural_event_file_name: ""

_id: ObjectId("5ec9be27f12aae959f9ef008")
prefix: "device"
device_type: 1
```

## User data:

The screenshot shows the MongoDB Compass interface. On the left sidebar, the 'users' collection is selected under the 'cs682' database. The main panel displays a single document from the 'users' collection. The document contains various fields including a unique identifier, user name, location coordinates, and sensor status flags.

```
{
  "_id": ObjectId("5ec9be27f12aae959f9ef1d3"),
  "prefix": "user",
  "selectedArea": true,
  "selectedLocation": true,
  "name": "User 1",
  "latitude1": 42.314360001377004,
  "latitude2": 42.31863191561976,
  "longitude1": -71.0368902618691,
  "longitude2": -71.04142681619297,
  "locationLongitude": -71.03890150854394,
  "locationLatitude": 42.311083106365245,
  "temperatureSensing": true,
  "humiditySensing": false,
  "gasSensing": false,
  "lightSensing": true,
  "windLevelSensing": true,
  "waterLevelSensing": true,
  "dataEncrypted": false,
  "preferredLatency": 10,
  "preferredThroughput": 0,
  "preferredFrequency": 0,
  "startTime": 0,
  "endTime": 64000
}
```

## Simulation Result data:

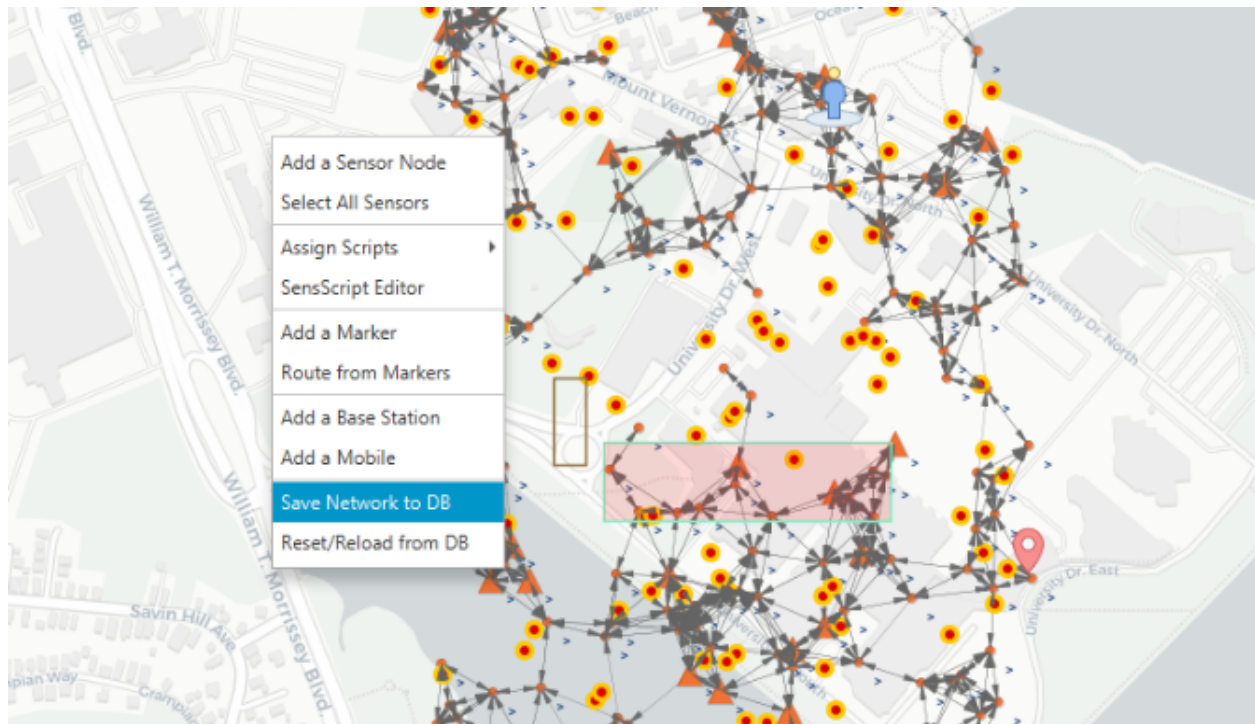
The screenshot shows the MongoDB Compass interface with the 'result' collection selected. The main panel displays three documents from the 'result' collection, each representing a simulation result. Each document includes fields for simulation time, sensor ID, sensor location, latency, wind level, and temperature.

```
{
  "Simulation time": "9.2648293333333342",
  "Sensor ID": 101,
  "Sensor location": "-71.04169567790824 42.31347825236258",
  "Lantency": " 0.0023460333333333336"
},
{
  "_id": ObjectId("5ec9765ef47ced53dcd1ae65"),
  "User ID": "User 1",
  "Simulation time": "11.453432000000015",
  "Sensor ID": 215,
  "Sensor location": "-71.0398952779279 42.31343274166158",
  "Lantency": " 0.0021886026666666733",
  "Wind Level": "20.24392104606157",
  "Temperature": "20.24392104606157"
},
{
  "_id": ObjectId("5ec97662f47ced53dcd1ae66"),
  "User ID": "User 1",
  "Simulation time": "14.051362666666673",
  "Sensor ID": 223,
  "Sensor location": "-71.04039419205354 42.31374812691422",
  "Lantency": " 0.0025979306666666577",
  "Wind Level": "16.13097419395484",
  "Temperature": "16.13097419395484"
}
```

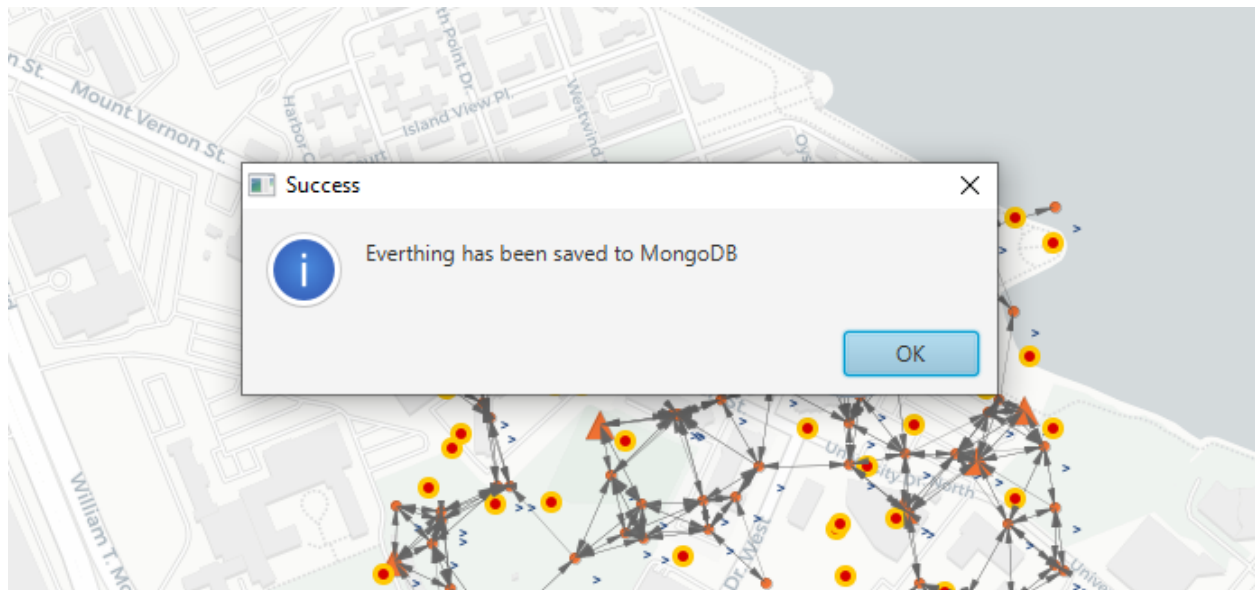
### 3.1. Import to Database

After creating a network and users, the user can save all current network data to MongoDB.

Right-click mouse and choose Save Network to DB:



Your network is successfully saved:

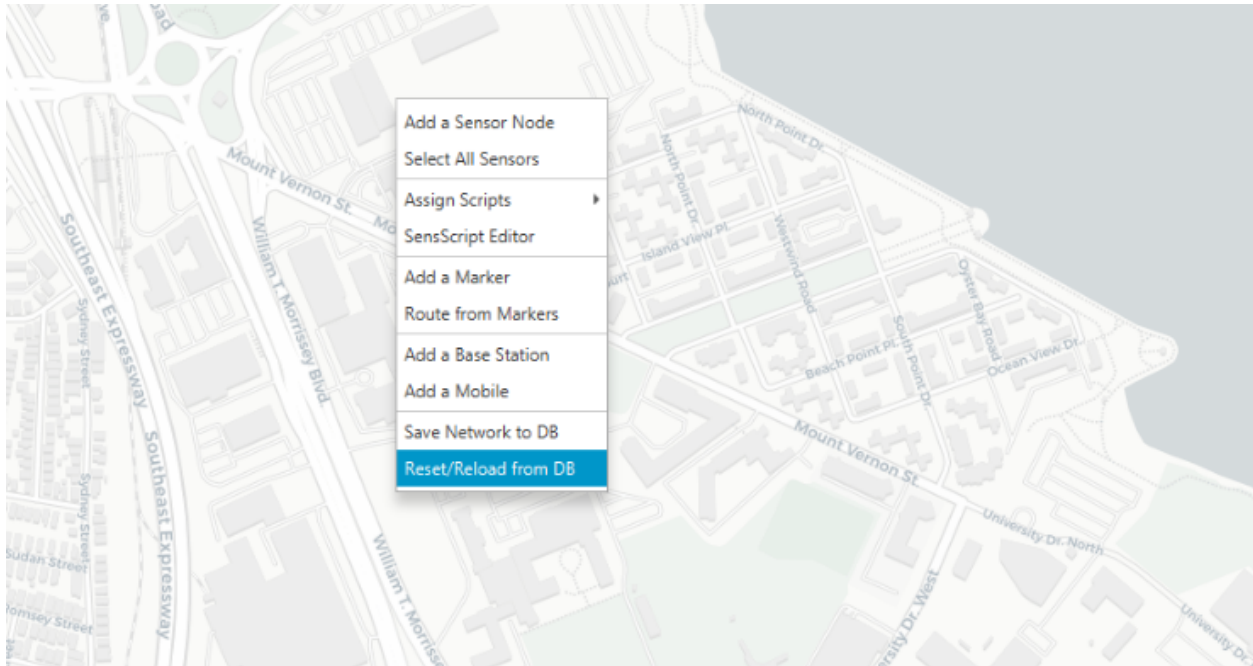




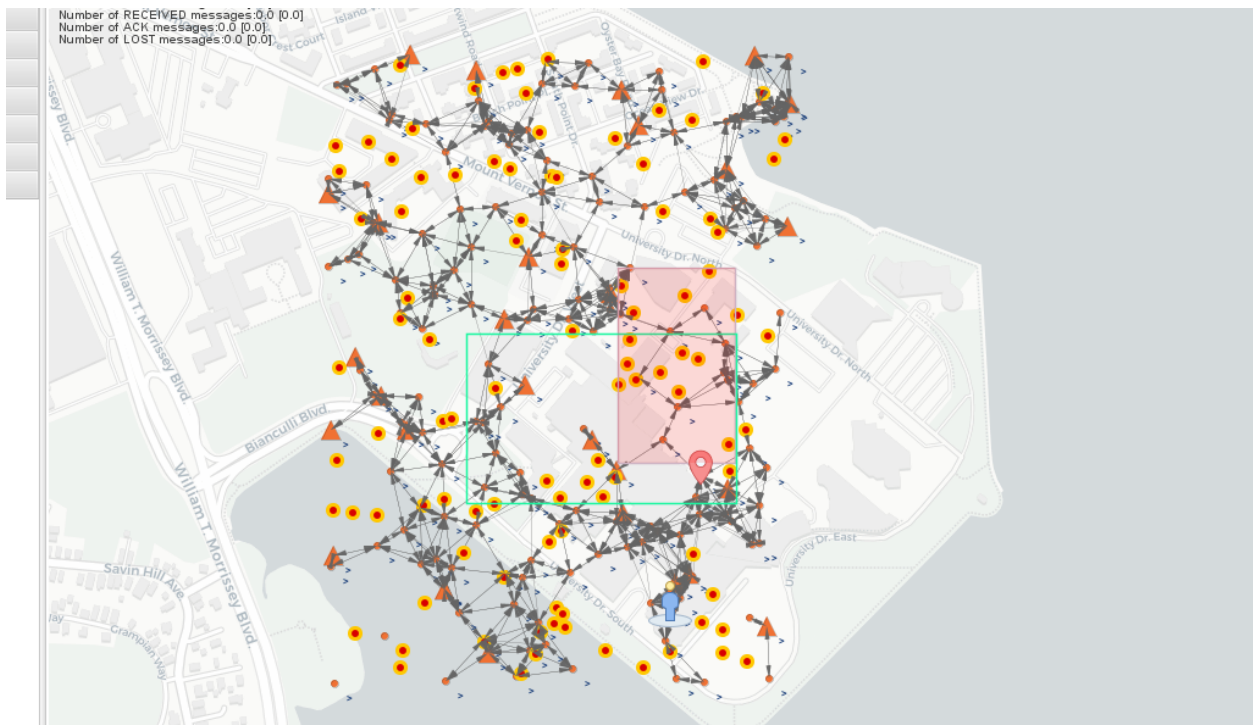
### 3.2. Export from Database

We can easily load data from our database.

Right-click mouse and choose Reset/Reload from DB:



Load network from database successfully:

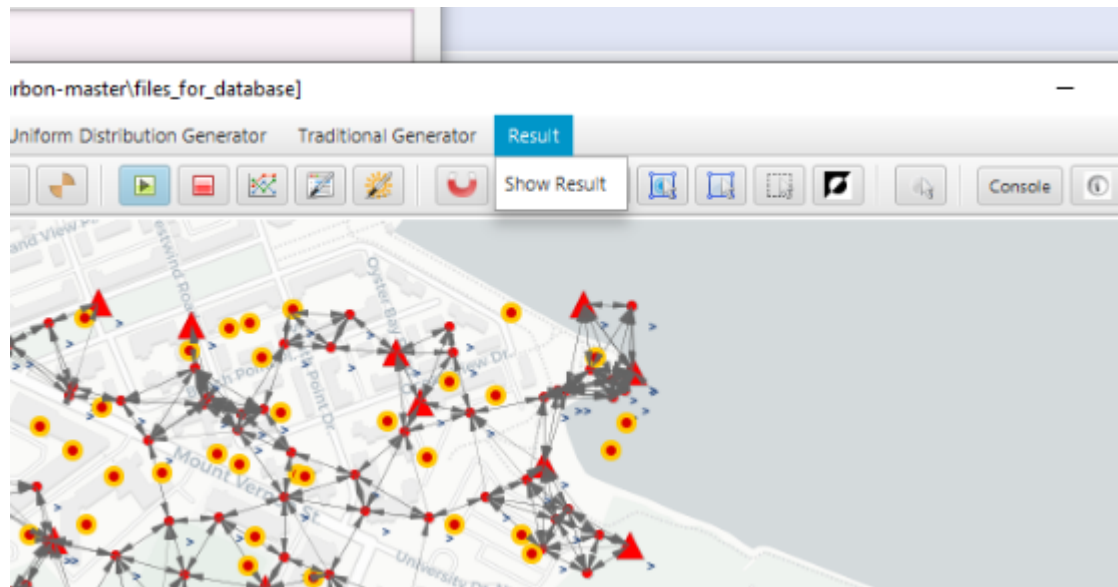


## 4. Result

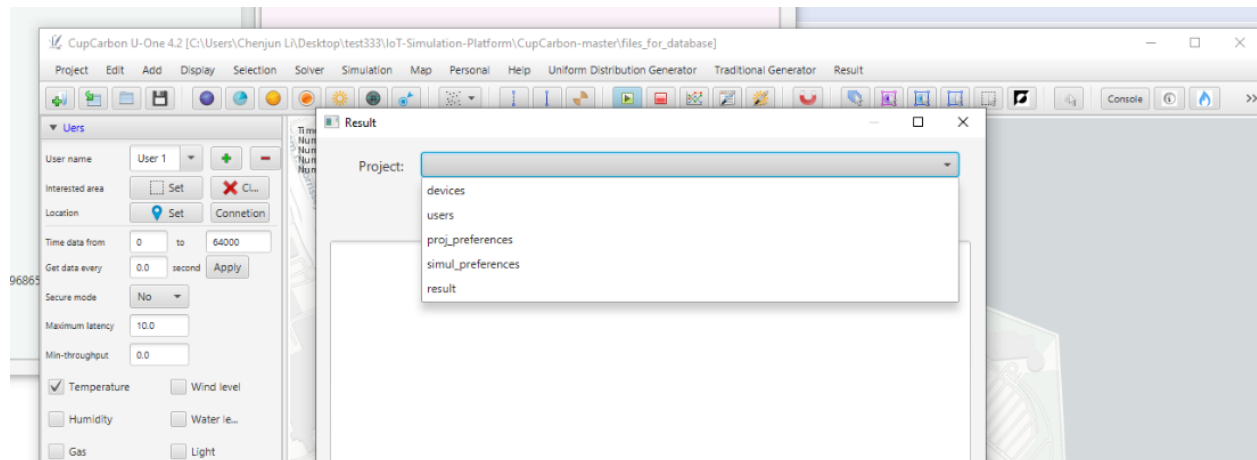
### 4.1. GUI Show Result

User can check all result data in the database from GUI.

Menubar->result->Show Result:

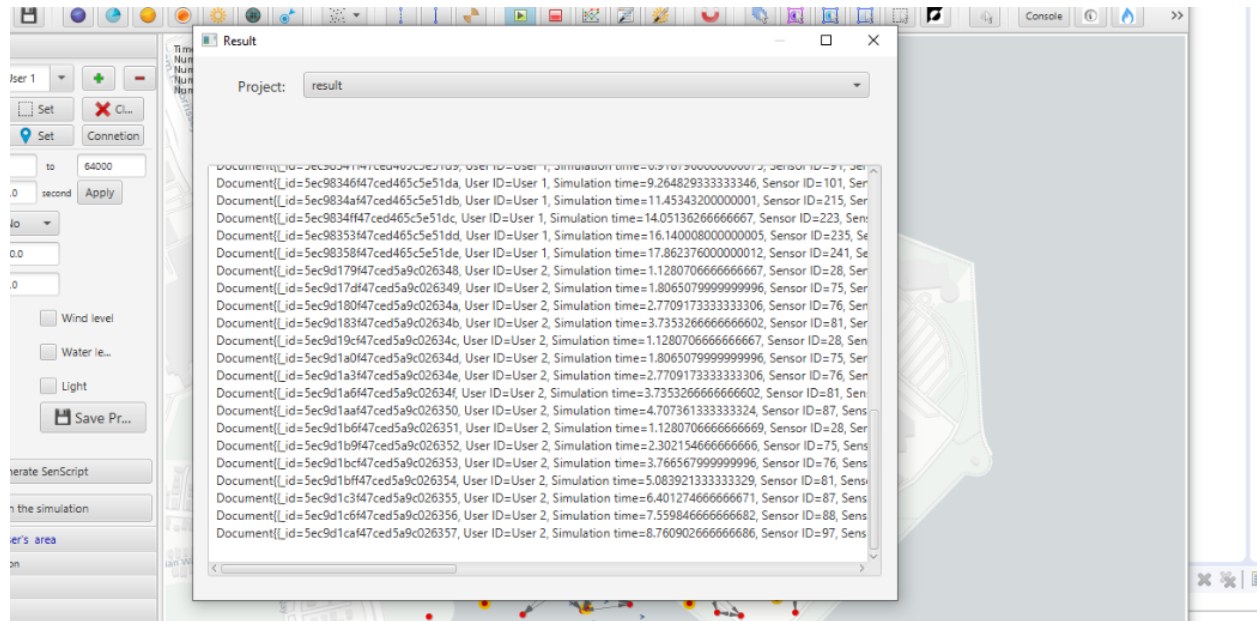


Select collection:





Show result:



## 4.2 printDB

The simulation result will be auto-saved into database by a customized SenScript(Check CupCarbon User Guide) command -- printDB.

The command printDB will read a generated special format data which including sensor id, sensor format, sensed types/values, latency, and simulation time. The command will also compare users' area of interests, and save only users' interests value to database. Latency here is different from latency in user's property. This latency is calculated by the time difference between the time user sends a request message and the time receives the message back.

The result format will be like this in JSON format:

```
{ User ID: user1,  
  Simulation Time: 2.3333333  
  Sensor ID: 256  
  Sensor Location: -28.3838388 71.900000003  
  Lantency: 0.00000003  
  Temperature: 25.66666  
  ...}
```

# 5. New Script Command

For simulation, we created some new script command and slightly edited some old command.

- AREADSENSOR
- LED
- GETU
- PRINTDB

## 5.1. AREADSENSOR

AREADSENSOR works slightly different than the old version.

AREADSENSOR X

It now returns a string in format below to X.

```
5#Temperature#25.333#Gas#84.333...#end
```

The first number is all nature events this sensor detected. Mostly one for each type, max number will be 6. The order of nature events is not guaranteed.

## 5.2. LED

LED works totally different than the old one since the old one is useless and will never be used.

LED V1 V2

V1 returns an array of all sensor ID which all connected users have an interest in.

V1[0] is the number of sensors.

V2 returns an array of all connected users' concerned sensing values.

V2[0] is the number of values.

## 5.3. GETU

GETU is a new command.

GETU X

GETU will generate a table which represents all connected users with its concerned area sensors, active/end time and latency.

X format:

5	X	X	X	X	X	X	X	X	X
8	user1	2.1	10.5	2	114	3	34	9	X
8	user1	0.1	22.5	1	134	5	34	22	X
7	user1	1.0	17.0	2	214	3	34	x	X
8	user1	0.0	100	0	204	9	14	2	X
9	user1	5.0	76.0	1	314	7	6	12	1

X[0][0] is the number of users.

X[i][0] where  $i > 0$  is the number of users concerned sensor + 5.

X[i][1] where  $i > 0$  is user name

X[i][2] where  $i > 0$  is user active time.

x[i][3] where  $i > 0$  is user end active time.

X[i][4] where  $i > 0$  is user prefer latency

X[i][j] where  $i > 0, j > 4$  is user's concerned sensor ID.

In the table, "X" means nothing, just padding for the table.

#### 5.4. PRINTDB

Check 4.2