

Master Thesis

**Household power consumption
monitoring and modeling**

Supervisor: Koji OKAMOTO, Prof

Meng LI

Dept. of Human and Engineered Environmental Studies
Graduate School of Frontier Sciences
The University of Tokyo

Acknowledgement

First and foremost, I would like to express my particular thanks to Prof. Koji OKAMOTO, who advised my work and thesis with great patience and has been taking care of me throughout the master course. Only with his enlightenment can I manage to complete the procedure, here I would like to present my highest appreciation for his kindness as well as his help.

I would like to express my sincere gratitude to those who are and who were with me both in the research and my daily life during the two years I spent here at Visualization Laboratory in Human and Engineered Environmental Studies, GSFS of the University of Tokyo.

I would also like to thank Satoshi SOMEYA-sensei, for his valuable suggestions and constructive comments on lab meetings.

A lot of contributive ideas in the discussion part were provided by Yasuhiro HASHIMOTO-sensei, his enthusiastic help is greatly appreciated.

Special thanks to my senior Mr. Takanori FUJIWARA, who worked on the same project, many ideas were produced in our discussions and cooperation.

Lastly I want to thank my friends including all my lab fellows (especially Yanrong LI-san, Naoki NISHIKAWA-kun, and Qiang XU-kun) who made my adventure in Japan a pleasant journey; and professors on lectures, secretaries (especially Noriko SHIMOKAWA-san), for their assists that made my school life an enjoyable one.

Table of contents

Acknowledgement	1
Table of contents	2
Chapter 1. Introduction and Objectives	4
1.1. Background	5
1.2. Necessity for a new simulation system	10
1.3. Objectives.....	11
Chapter 2. Methodology	12
2.1. Power monitoring system.....	13
2.1.1. Overview	13
2.1.2. Hardware components	13
2.1.2.1. Metering devices.....	14
2.1.2.2. Raw data gatherer.....	16
2.1.2.3. Database Server	17
2.1.3. Software components	18
2.1.3.1. Work flow of PMS	19
2.1.3.2. Table design.....	19
2.1.3.3. Modules and file structure	21
2.1.4. Result	26
2.2. Simulation environment	32
2.2.1. Overview	32
2.2.2. Appliance simulators.....	34
2.2.2.1. State machine in modeling appliance.....	34
2.2.2.2. Non-interactive and interactive appliances	35
2.2.2.3. Permissions on interactive appliances	36
2.2.2.4. Power consumption functions	37
2.2.2.5. Control logic of appliance simulator	38
2.2.2.6. Parameters in profiles	40
2.2.2.7. Profiling of appliances	42
2.2.3. Agents	44

2.2.3.1. Layer structure in modeling user behavior.....	44
2.2.3.2. Randomness in human behavior	45
2.2.3.3. Control logic of agent simulator.....	46
2.2.3.4. Parameters in profiles	47
2.2.3.5. Profiling for agent.....	49
2.2.4. Implementation	50
2.2.5. Result	52
Chapter 3. Discussion	55
3.1. Application of the simulation environment.....	56
3.2. Policy design on supplier end.....	57
3.2.1. Existing pricing policies	57
3.2.2. Pricing function in smart grid	63
3.3. Task scheduling mechanism on appliance end	65
3.3.1. Non-emergent tasks and background usage	65
3.3.2. Schedule algorithm for non-emergent tasks	67
3.4. Test examples	70
3.4.1. Environment setting	70
3.4.2. Simulation result and discussion	72
3.4.2.1. Test of schedulable tasks in different quantity.....	72
3.4.2.2. Test of price stabilizing effect	75
3.4.2.3. Test of effective conditions	78
3.5. Summary	83
Chapter 4. Conclusion	84
4.1. Research summary	85
4.2. Conclusion.....	86
4.3. Future work	87
Chapter 5. References	88
Chapter 6. Appendix	90
Part A: Source code of PMS	92
Part B: Source code of SE.....	102
Part C: the monitored appliances are:.....	125
Part D: template profiles for each appliance:.....	132

Chapter 1. Introduction and Objectives

1.1. Background

As electronics continuously come to our daily life in an ever increasing number, and their usage become an indispensable part of our working and life experience, the demand for a reliable, and capacious power supply is being paid more and more attention. In modern power grid, the typical configuration can be illustrated as in figure 1.1:

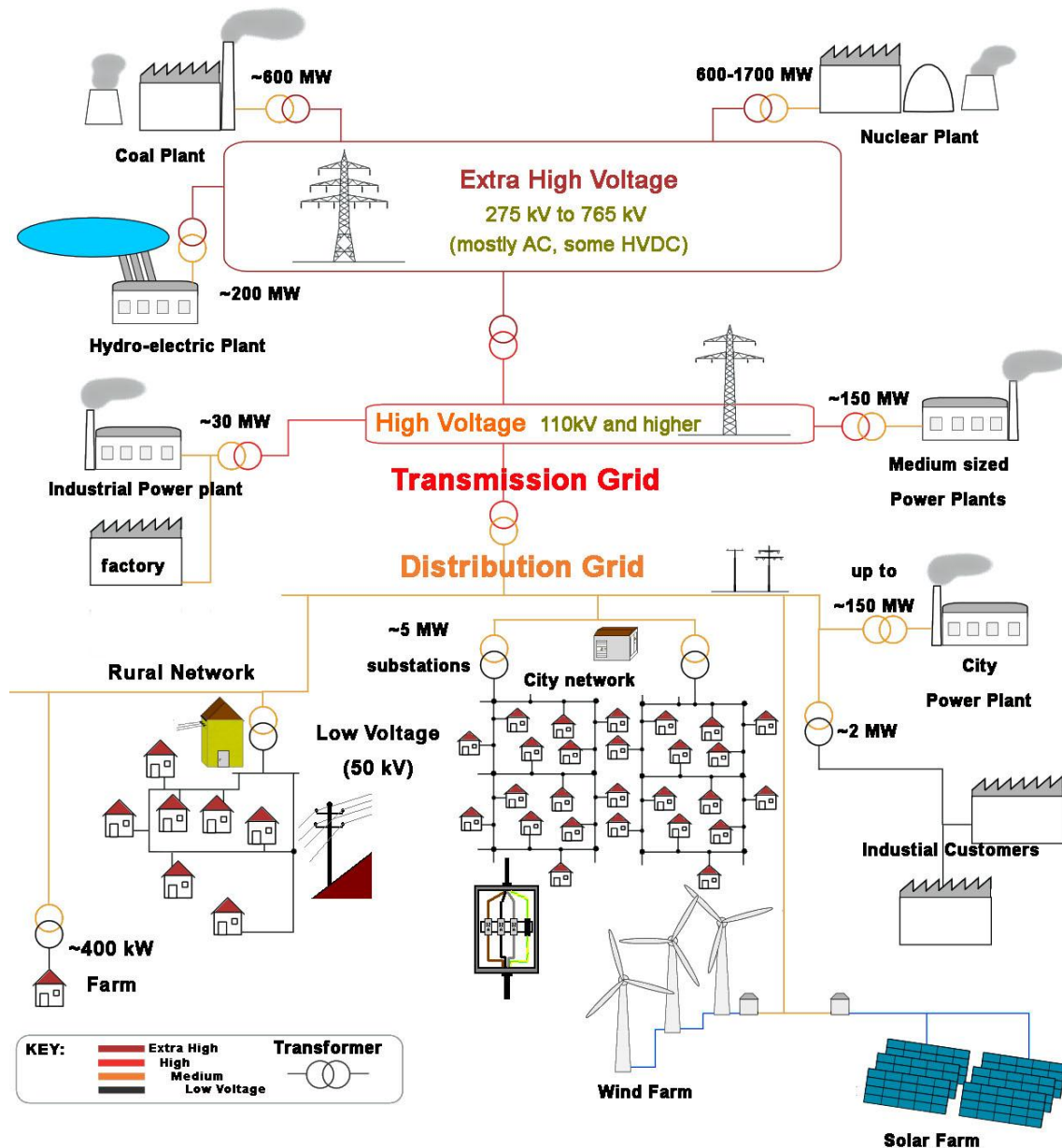


Fig.1.1. Electricity grid schematic

(Image source: Electricity grid schema by Stefan Riepl (originally in German, English version by J JMesserly), Wikimedia foundation, 2008)

In this setup, the controlling is all centralized in the power generation end, power supplier runs the supervisory control and data acquisition (SCADA) and generate power according to usage estimation and system feedback, the generation is demand following.

There are few problems with this schematic, one key example is that, the grid AC frequency may drift up or down from the nominal value. The grid has an effective physical rotating inertia from the combined total of all the rotating generators connected to the grid. If more power is coming in from generation than going out to loads, there will be a net torque on the system that causes all the generators to speed up, increasing the grid frequency [1]. Figure 1.2 shows a sample time history of the frequency on the grid in the western US, sampled six times a second. The slope of the frequency trace is a measure of the overall imbalance of generation and load at any given moment.

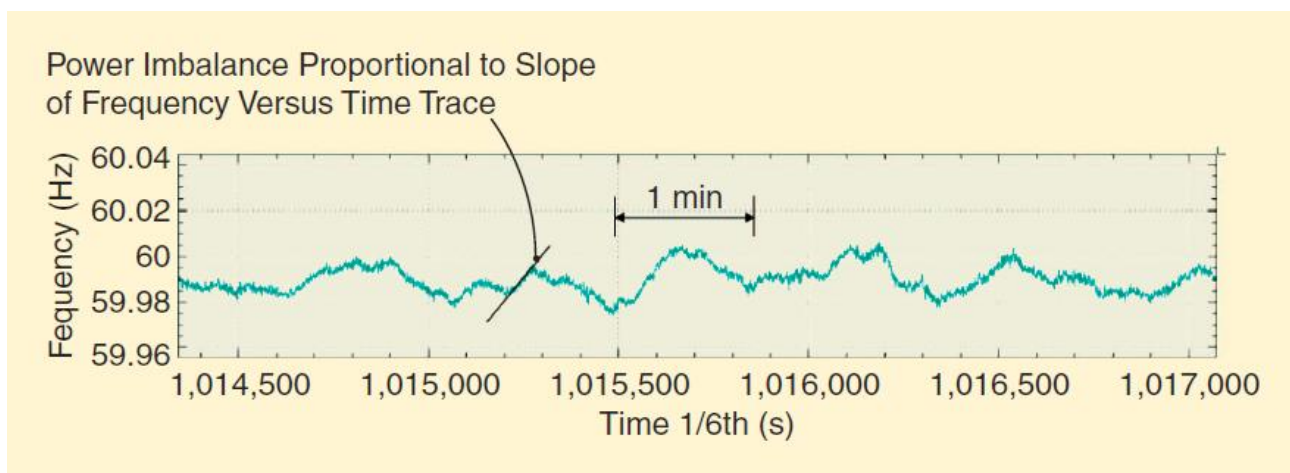


Fig.1.2. Grid power imbalance causes changes in frequency

On the other hand, with depletion problem of fossil fuels and public's worrying about nuclear power, the popularization of renewable energy resources is emerging. However, the new devices' entering, such as household solar panels, small scale wind turbines, could disastrously magnify the instable effect, because the net energy consumption will be more unpredictable, which makes it more difficult to regulate the grid with demand following.

The nature of renewable energy resources decides their output quite unstable compares to traditional coal and nuclear power generation. For example, wind generation profiles are different every day, with the time of day of peak generation varying by many hours from day to day [1]. Figure 1.3 is one month's worth of daily wind generation profiles in the PG&E territory.

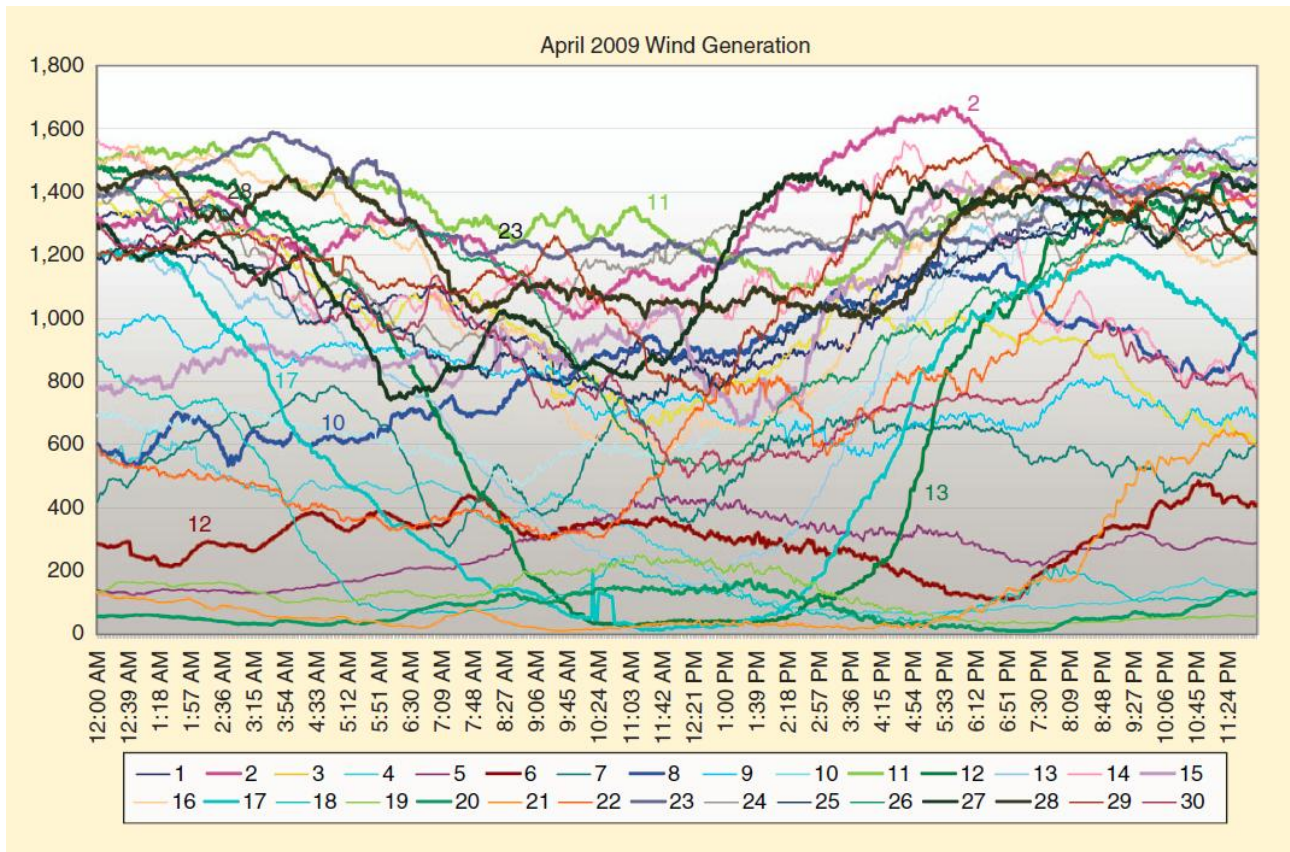


Fig.1.3. Tehachapi California wind production for each day of April 2009.
(Image courtesy of Cal ISO.)

The electric power industry in the 21st Century will see dramatic changes in both its physical infrastructure and its control and communication infrastructure. These changes are the result of mainly three factors [2]:

1. The push toward a deregulated industry
2. The development of more efficient and/or less polluting energy resources
3. The continued electrification and integration of information technology into the most facets of our everyday lives

Besides of these, traditional SCADA is facing challenges, as people require the following features in the next generation power grid [3]:

1. Better reliability
2. High quality power supply
3. Integration with more distributed energy resources (DERs)
4. Consumer participation
5. Optimized assets

Smart grid is proposed as whole solution to meet these requirements. The term “Smart grid” has been in use since at least 2005, when “Toward A Smart Grid”, authored by S. Massoud Amin and Bruce F. Wollenberg appeared in the September/October issue of IEEE P&E Magazine(Vol. 3, No.3, p34-41).

A smart grid delivers electricity from supplier to consumers using two-way digital technology to control appliances at consumers' home to save energy, reduce cost and increase reliability and transparency [3]. The electricity distribution grid is overlaid with an information and net metering system that keeps track of all electricity flowing in the grid. The grid also incorporates the capability of integration DER by utilizing local power storages [5][6]. The conceptual formation is shown in the following figure 1.4:

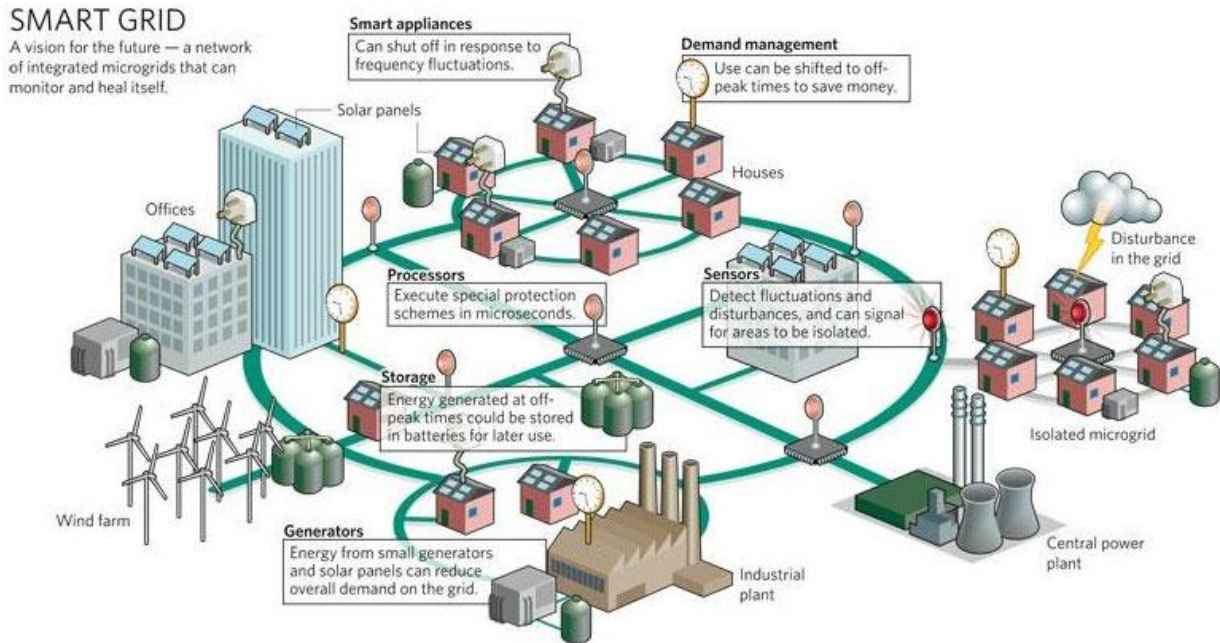


Fig.1.4. Conceptual schematic of smart grid with mechanisms
The Smart Grid Frontier: Wide Open, by David Heyerman
(Image source: tinycomb.com, 2009)

By utilizing modern communication technologies, dynamic regulating mechanism becomes more effective in smart grid, with reliable failure resisting mechanism. Integration with DER and data transmission mechanism will enable power market [4][7], a new feature in future power grid. Both passive demand following and active demand controlling will be applied in smart grid for making regulating tasks easier to achieve. Floating price feature will be introduced to help individual consumer in saving costs on the power market and help power supplier in managing customer consumption of electricity in response to supply conditions, so the desired behavior of the whole power grid can be achieved, for example, having electricity customers reduce their consumption at critical times or in response to market prices.

In smart grid, consumers participate actively to save cost in the meanwhile of optimizing the usage in the whole grid. It is achieved by allowing appliances and DERs in the grid respond to signals broadcasted by the supplier, which can be controlling signals and/or power price, as shown in figure 1.5.

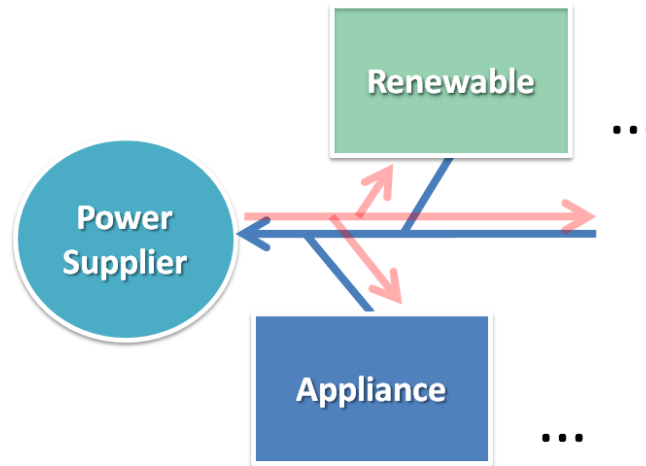


Fig.1.5. Signal broadcasting and usage control

In this configuration, individual consumers are viewed as independent entities with accessing to power grid features such as power supplying and signal receiving. A number of consumer and DERs form a small scale grid that integrates local controlling, optimization, which is defined as micro-grid. The controlling in future smart grid is majorly implemented at this level [5]. As power consumers in micro-grid are independent to each other (though more or less information exchanging is designed for achieving individual's usage strategy as well as a greater goal such as usage balancing), and distributed control is used, the micro-grid control can be described and researched as a multi-agent system controlling problem [5][6][7][8].

Demand response (DR) respond to explicit requests to shut off appliances, whereas power price is just a moderate controlling encourages/discourages appliances to manage usage according to their own strategies [13].

Floating price is another way of realizing usage control in a power grid [14][15]. Unlike in DR, the mandatory controlling signal is directly sent to the appliances, floating price is just a moderate approach encouraging usage scheduling at individual consumer's, utilizing people's concern about saving power cost.

1.2. Necessity for a new simulation system

With the radical changes that taking place in the power grid, it is important to build a load simulation system for:

1. Prototyping smart grid for new regulating mechanisms
2. Validating the usefulness of smart appliances and DERs
3. Validating the effectiveness of different policy design

Existing simulation systems built based on statistics of historical consumption data are lack of credit when considering the new changes that would happen in the future power grid. Future changes would happen not only in the scale but also in the composition of electricity infrastructure. Appliances such as electric vehicles and local power storage are yet uncertain of their behaviors; Types of DER are coming to our life with different quantities and capabilities.

Therefore, experiential model is not competent to make accurate load estimation and is not flexible in modifying the composition.

It is necessary to build a new simulation system that can do the work of load estimation with integration of new appliances and DERs, we proposed a multi-agent system approach in build such a simulation environment, which is flexible to change the composition at a resolution of single device, as well as to adjust the scale from one single room to tens of thousands of rooms. The simulation environment can be used not only as a power forecasting tool with improved resolution but also the testing and verifying platform of new policies and strategy design on the large-scale. We expect the system to be a useful tool to aid future power grid design and evaluation of smart grid features.

1.3. Objectives

This research aims for providing a useful tool in designing future smart grid and eventually facilitates building a more robust and intelligent power grid that makes people's life more convenient and efficient.

In this paper, we examined the method of multi-agent system (MAS) in building a load simulation environment (SE). We utilized advanced metering devices (AMI) and readymade information system in building a power monitoring system (PMS), and based on the practical data captured by PMS, we built the SE.

On the system application layer, we discussed the advantage of bringing in micro-scale controlling algorithm on smart appliances, and examine its effectiveness on SE. The overall process is shown in the figure 1.6:

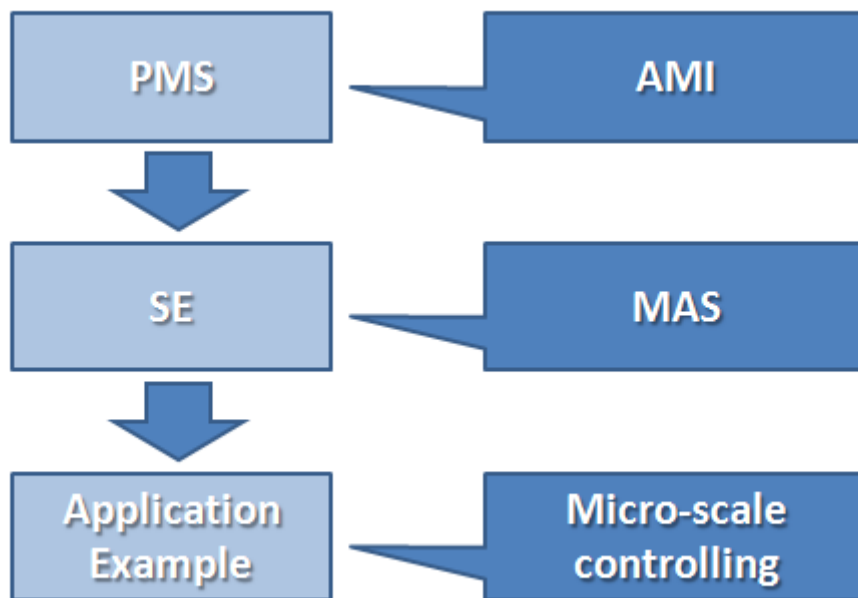


Fig.1.6. Overview of the research, processes and objectives

Chapter 2. Methodology

2.1. Power monitoring system

2.1.1. Overview

In this part, we implemented the power monitoring system (PMS) based on ready-made smart metering system, in order to get practical data for next step appliance modeling.

2.1.2. Hardware components

We use smart metering system provided by Eicoh (www.eicoh.com), which includes smart meters and a raw data gatherer. The metering devices are used for measuring the current power of appliances plugged in, and send the power measures to the raw data gatherer through wireless network, as shown in figure 2.1. We built up the database server to retrieving raw data files from raw data gather, the data records are stored in the database for later inquiries.

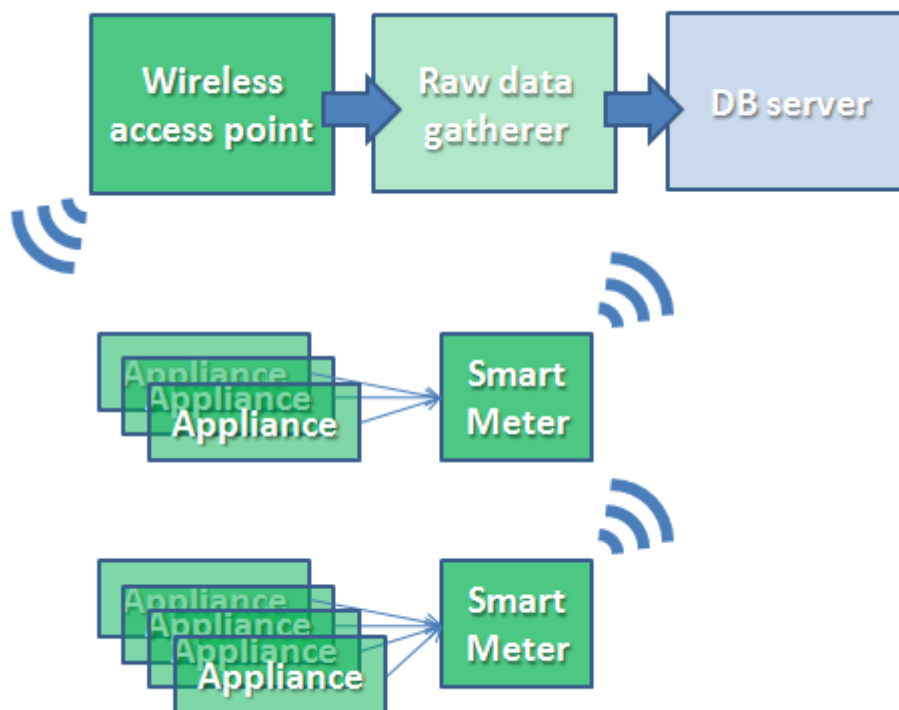


Fig.2.1. Hardware components in PMS

2.1.2.1. Metering devices

There are two models of meter in our experiment setup: 3-slot model and 4-slot meter. Compares to the 4-slot model, 3-slot model has fewer slots or plugging with appliances but with an extra temperature and humidity sensor built in. Figure 2.2 shows the picture of 3-slot (left) and 4-slot (right)



Fig.2.2. Smart metering devices

The maximum current should be limited to 15A. A breaker of 20A is built in for security considerations.

When the meter is powered, it automatically starts sending data after 10-13 seconds (to avoid concurrent conflict in network), the interval of data sending can be set from 1-10 minutes. In our experiment, this interval is set to 10 minutes for releasing the load in wireless network. When the interval is set to 0, the meter will stop sending power measures.

The statuses of the meter are: setting wireless module, connecting, sending data, an indicator near the antenna shows the current status of the meter:

Indicator	Status
Switching (Irregular interval)	Setting wireless module
Switching (Regular interval)	Connecting
On	Sending data

Table.2.1. Correspondence between indicator and meter status

When setting up the device, software driver provided by Eicoh is needed to be installed on a Windows PC. Then connect the meter and PC with a USB cable and set via command lines on terminal software (we used TeraTerm), as shown example in figure 2.3, the commands are listed in table 2.2. Settings are saved when the meter is power off.

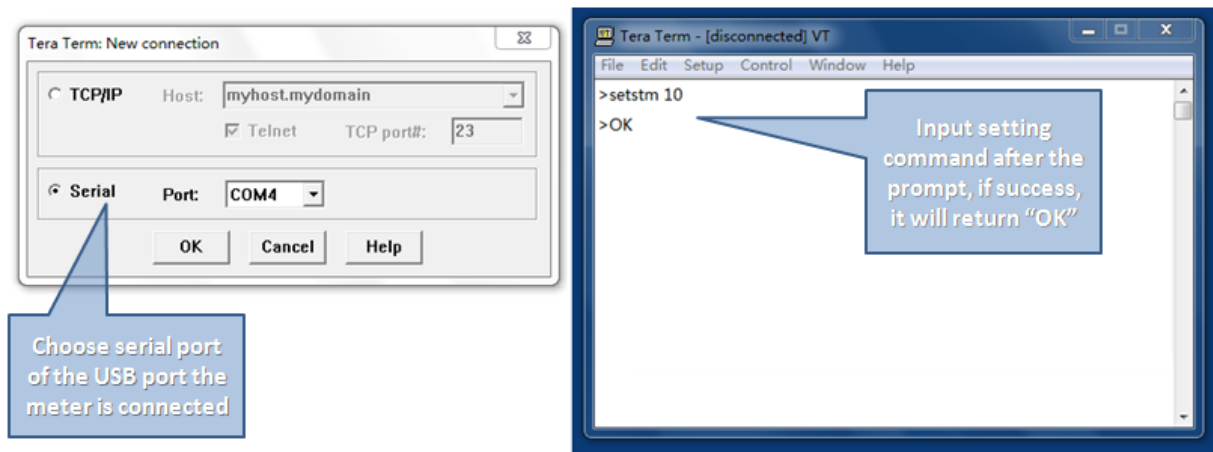


Fig.2.3. Example of setting up a meter on terminal software.

Function	Command
IP address setting	setip XXX.XXX.XXX.XXX
Subnet mask setting	setmask XXX.XXX.XXX.XXX
Gateway setting	setgate XXX.XXX.XXX.XXX
Server IP address setting	setsv XXX.XXX.XXX.XXX
Server port setting	setport n
DHCP setting	dhcp on/off
Password mode setting	smode wep/wpa
SSID setting	ssid s
WEP key setting (characters)	wep s
WEP key setting (hexadecimal)	weph hh
WPA key setting (characters)	wpa s
WPA key setting (hexadecimal)	wpah hh
Authorization mode (0:open/1:shared)	auth 0/1
Data sending interval setting (0:stop/1-10:mins)	setstm n
Display settings	adsp
Display firmware version	ver
Display help	help/?

Table.2.2. List of setting command

2.1.2.2. Raw data gatherer

The raw data gatherer is used to gather the power measure from the meters, figure 2.4 shows the picture of raw data gatherer and wireless network access point:

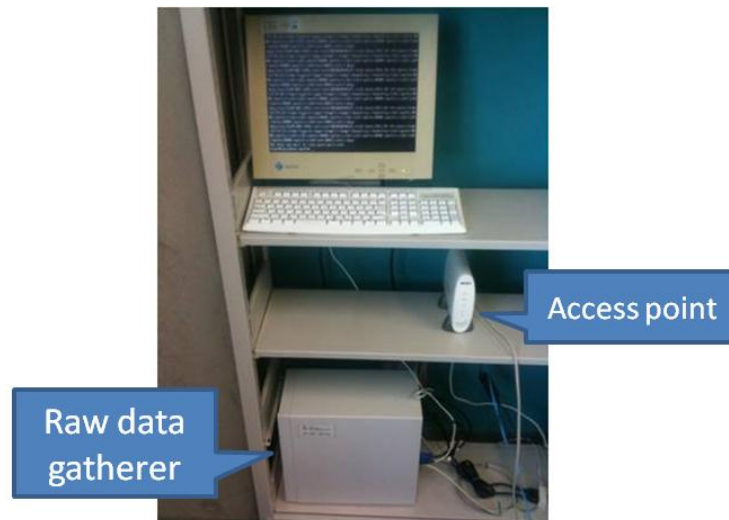


Fig.2.4. Raw data gatherer and wireless network access point

The raw data gatherer is running CentOS 5 Linux distribution, with Apache HTTP server and PHP runtime. On the raw data gatherer, directory and file structure is organized as shown in the figure 2.5:

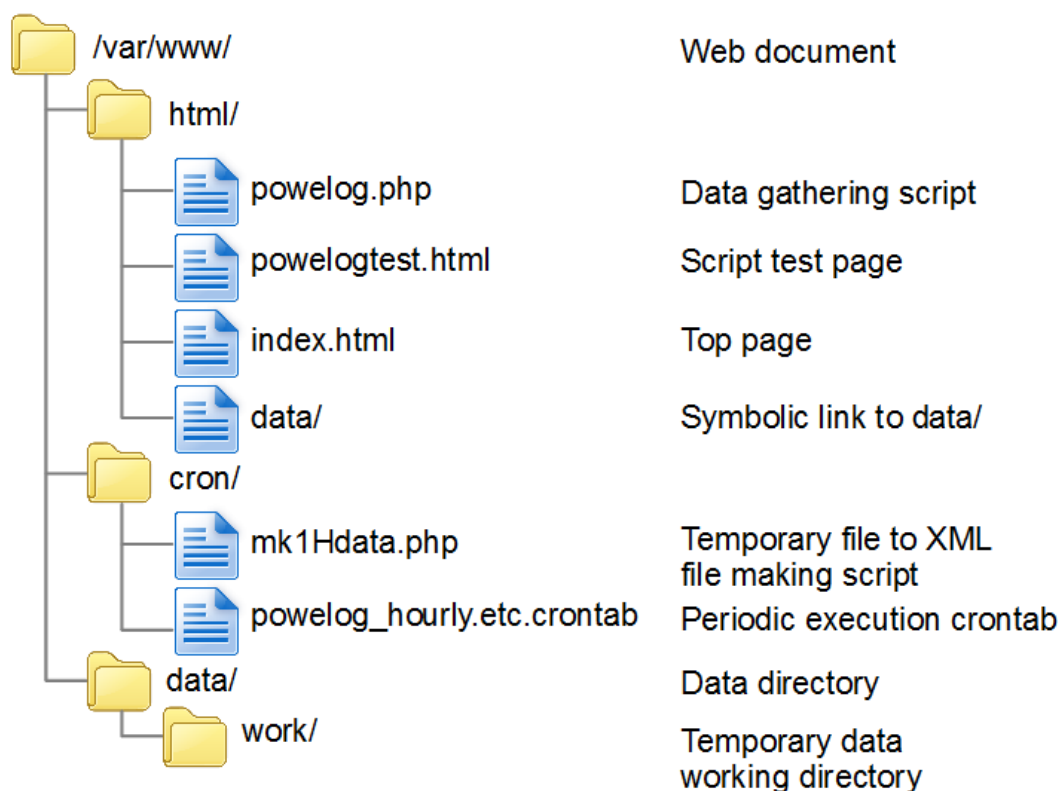


Fig.2.5. Directory and file structure on raw data gatherer

Temporary data file in work directory is updated whenever measures come from the meters, and mk1Hdata.php is scheduled to run every hour and write a XML file from these temporary files and delete the source temporary files. The XML files are named in the following format: YYYYMMDD-hh.xml, the file is contained in the directory of the following path: /var/www/data/YYYY/YYYYMM/YYYYMMDD/

The XML record file gathered this way is of the format:

```
<powerlog>
  <msg>
    <src>(Device model)</src>
    <uid>(MAC address)</uid>
    <date>YYYY-MM-DD</date>
    <time>hh:mm:ss</time>
    <type>(Device type)</type>
    <tempC>(temperature)</tempC> #valid on 3-slot version only
    <humidity>(humidity)</humidity> #valid on 3-slot version only
    <ch1><watts>XXXX</watts></ch1>
    <ch2><watts>XXXX</watts></ch2>
    <ch3><watts>XXXX</watts></ch3>
    <ch4><watts>XXXX</watts></ch4> #valid on 4-slot version only
    <ext></ext>
  </msg>
  ...
</powerlog>
```

As the raw data gather is only for storing raw power records, querying single or multiple appliance's power over time is difficult, so we build the database for all the appliances measured this way, in order to easily get the power-time chart for later analysis.

2.1.2.3. Database Server

The database server stores a duplication of power data in an organized way, and periodically synchronizes power data with raw data gather. It is based on the consideration that the crash of database server would not affect the original data. The database server is also a web server for querying the database and returning the result to user's browser window.



Fig.2.6. Database server, Dell PowerEdge T310

2.1.3. Software components

We use the popular “LAMP” environment for the database server setup: Red Hat Enterprise Linux 5 for the L, Apache HTTP server for the A, MySQL database manages system for the M, and Python scripts for the P.

We choose Python as the server end scripting language, as it is a powerful, general-purpose high-level programming language, especially suitable for fast prototyping. Its design philosophy emphasizes code readability, which is believed to be beneficial for further development even by different group of programmers. What more is that it has a large number of free and comprehensive third party libraries, which enables us to do more work with less coding.

2.1.3.1. Work flow of PMS

The work flow in the database server is shown in the figure 2.7:

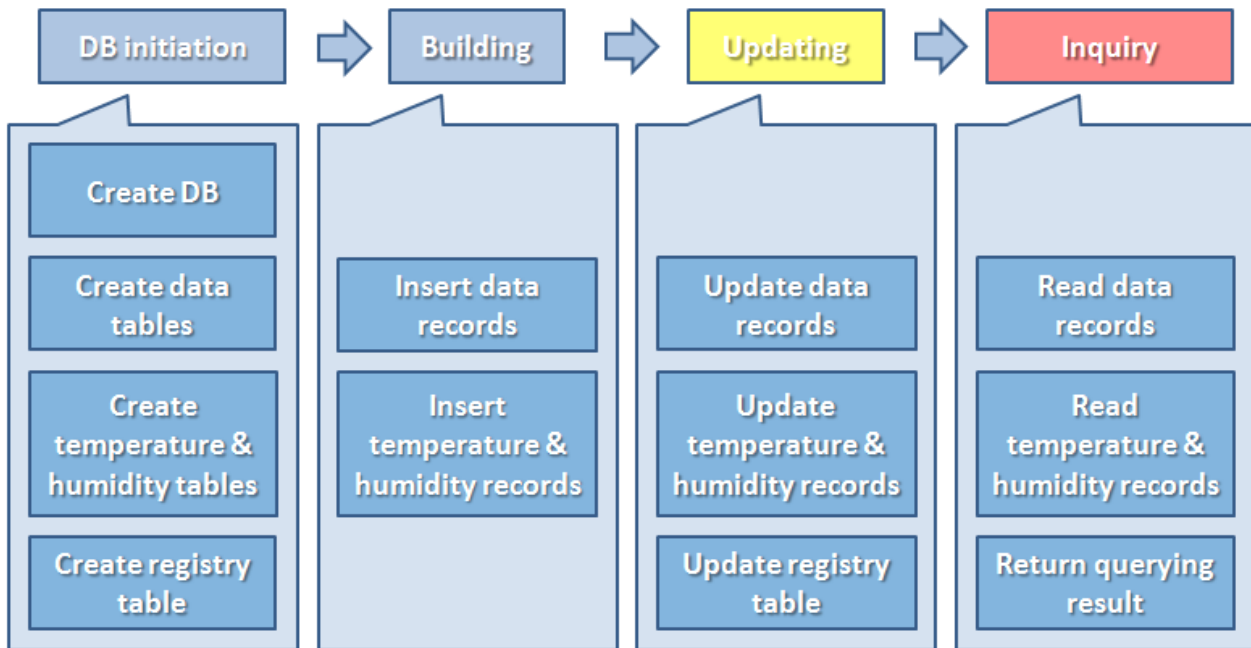


Fig.2.7. Work flow in database server

The database initiation is executed only once at installing the system. When building the system after the initiation, we specify the start time and run the building module only once. After the database is built up, updating is performed automatically once every day to write the data of the previous day into the database. The inquiry may happen several times a day depending to users' requests.

2.1.3.2. Table design

As the core part of database designing, the tables are designed according to the following guidelines:

1. Explicit and comprehensive form
2. Good performance in updating and querying
3. Acceptable size of storage

The raw XML file on data gather server contains the messages from all the meters within one hour in the reception order. The meter and time order may slightly vary depending on the network condition, as shown in figure 2.8. So for the sake of explicit format, we parse the XML file into message groups according to the meter identifier, which is its hardware MAC address. Within the same message group, messages are sorted in the time order.

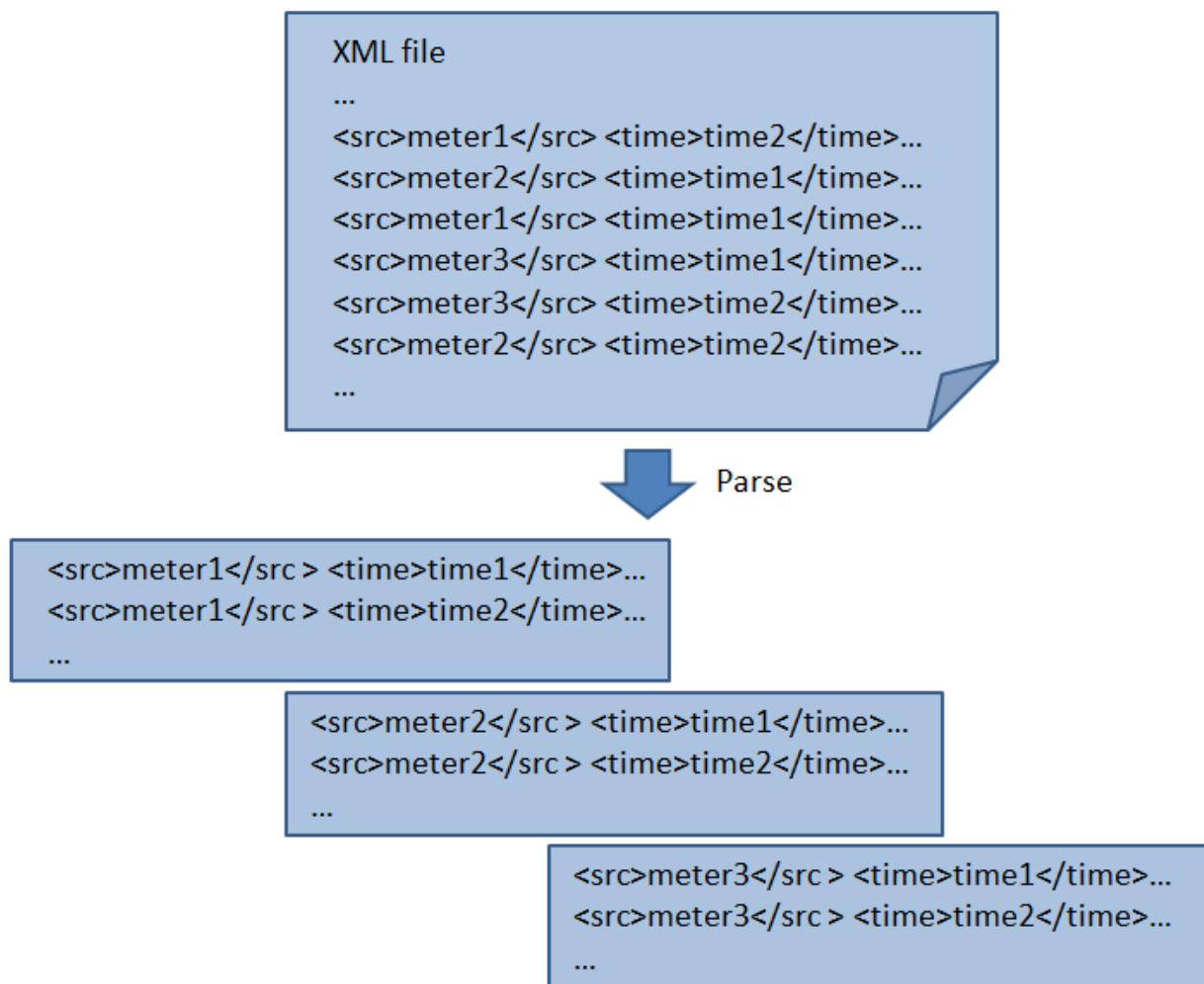


Fig.2.8. XML data file parsed into message groups

Each piece of message contains the following information:

1. Meter ID
2. Time stamp
3. Power measures for channel 1,2,3,4 (channel 4 is valid only on 4-slot version)
4. Temperature measure (valid only on 3-slot version)
5. Humidity measure (valid only on 3-slot version)

We made 4 types of tables in our PMS, the power data table, temperature data table and humidity data table, and the registry table.

In the power data table, there are two columns: time stamp (date and time) and power, the type of time stamp is the datetime type provided in MySQL DBMS, whereas the power is of integer type, the unit is Watt. As there may be multiple power measures for different channels/appliances in one piece of message, for convenience, we make a table for each channel/appliance; figure 2.9 (left) shows the procedure.

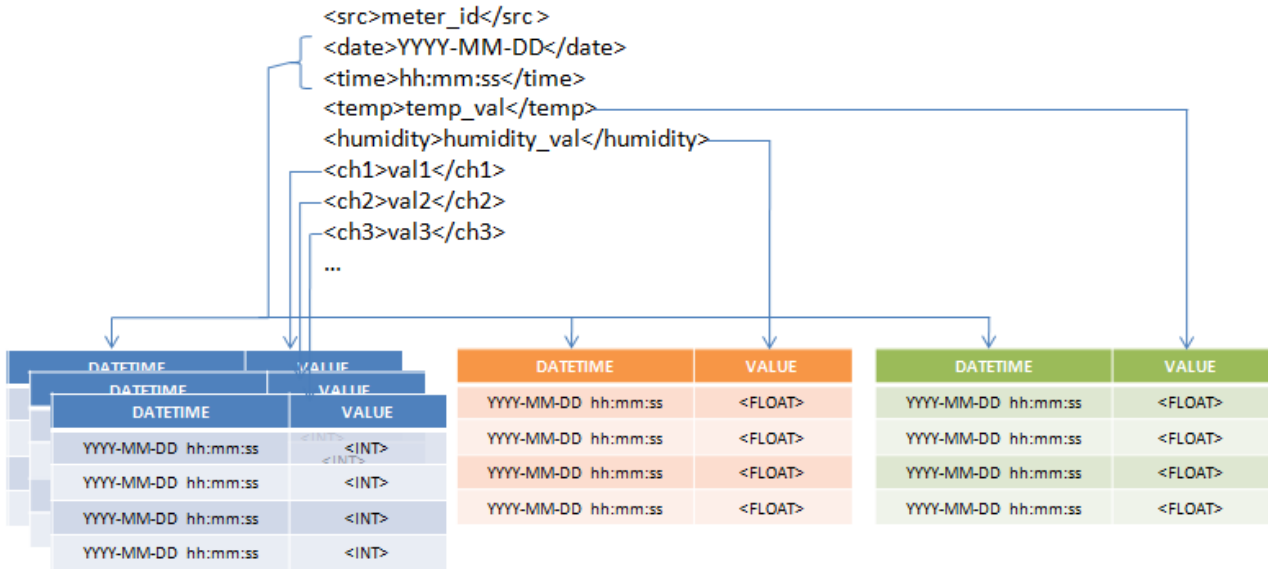


Fig. 2.9. Message updating into: power data table (left), temperature table (center), humidity table (right).

The temperature and humidity table are of similar design to power data table, the value is also integer in order to do fast access.

Besides the data tables, a registry table is also maintained in the DBMS, the registry is for storing the information of appliances plugged in, the columns in the table are: mac_address, location, channel1, channel2, channel3, channel4. Channel 4 is left blank if the meter is of a 3-slot type.

2.1.3.3. Modules and file structure

The building of PMS includes MySQL DBMS and modules programmed in python. We use MySQLdb, a python library to access MySQL from python programs; the python modules perform tasks such as db creating, table creating, raw XML file parsing and table updating.

On the querying part, we use inquiry module through CGI on the Apache HTTP server. The configuration of modules is shown in the figure 2.10.

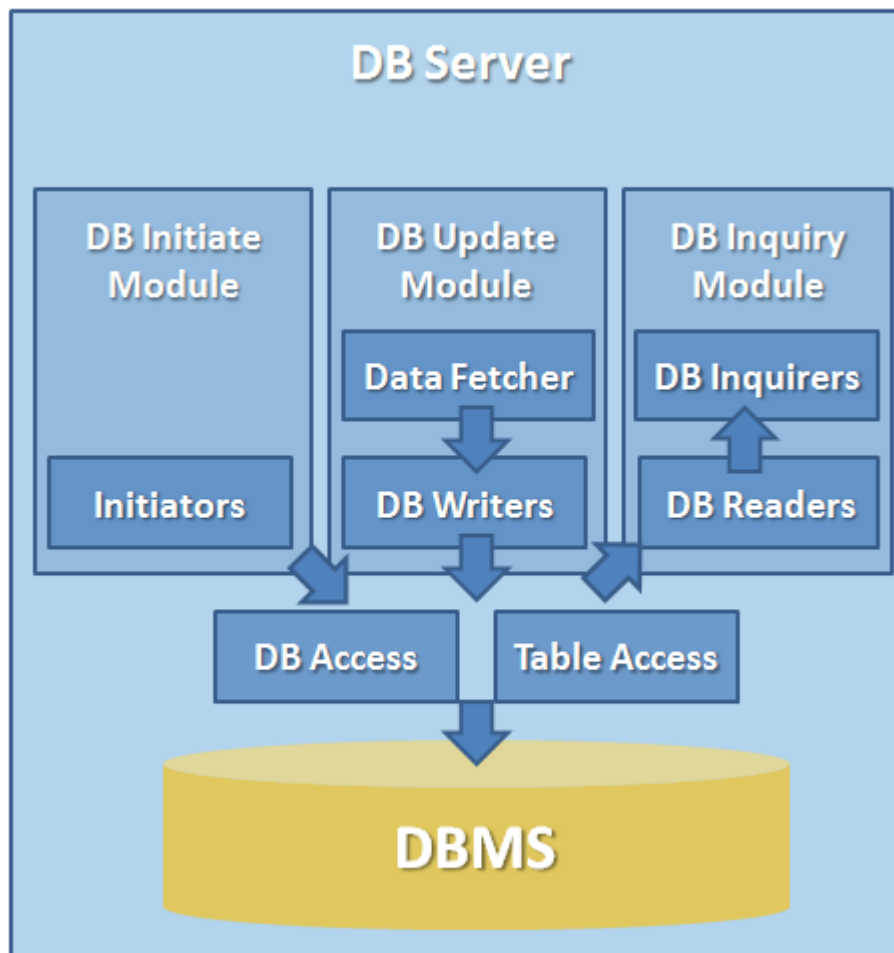


Fig.2.10. Python modules in PMS

The initiators in initiate module are used for create database and then in that database, create tables for power, temperature, humidity data and registry. The tables are empty after the creation; the building procedure is needed, which involves using the update module.

The update module performs both tasks for building the database for the first time run, and its periodical updating. The periodical updating is scheduled to run at 5:00 AM every day, the XML raw files gathered from 01:00:00 in the previous day to 03:00:00 in the current day, are retrieved and written into database after parse. Note the raw gatherer system may need some time to make the XML file, so some time should be allowed to wait for it finishes the task, little overlap in the period is designed to avoid data loss. The tables for power, temperature and humidity data are updated in this procedure. The registry table is not automatically updated with other tables; therefore, when building the table for the first run or the setup changes, it needs to be manually modified. The periodical updating is achieved by the cron job-scheduler on Linux, the crontab for doing this is written as:

```
0 5 * * * /home/user1/PMS/update/update_tables_data.py
0 5 * * * /home/user1/PMS/update/update_tables_temp_humidity.py
```

(The PMS folder is placed in the user's home folder in this example, all the files' permission should be set to free access for everyone so as the system can use them)

The inquiry module is used for querying the database and writing the results into a local CSV format file, or sending the results to a remote browser who requested the inquiry. The files are placed in the path: /var/www/cgi-bin/. (This module may be invoked by system; all the files' permission should be set to free access for everyone.)

The file structure of modules is shown in the figure 2.11 and 2.12, the call relations among the files, with functions noted, are shown in figure 2.13, 2.14 and 2.15. Detailed function descriptions are listed in the table 2.3.

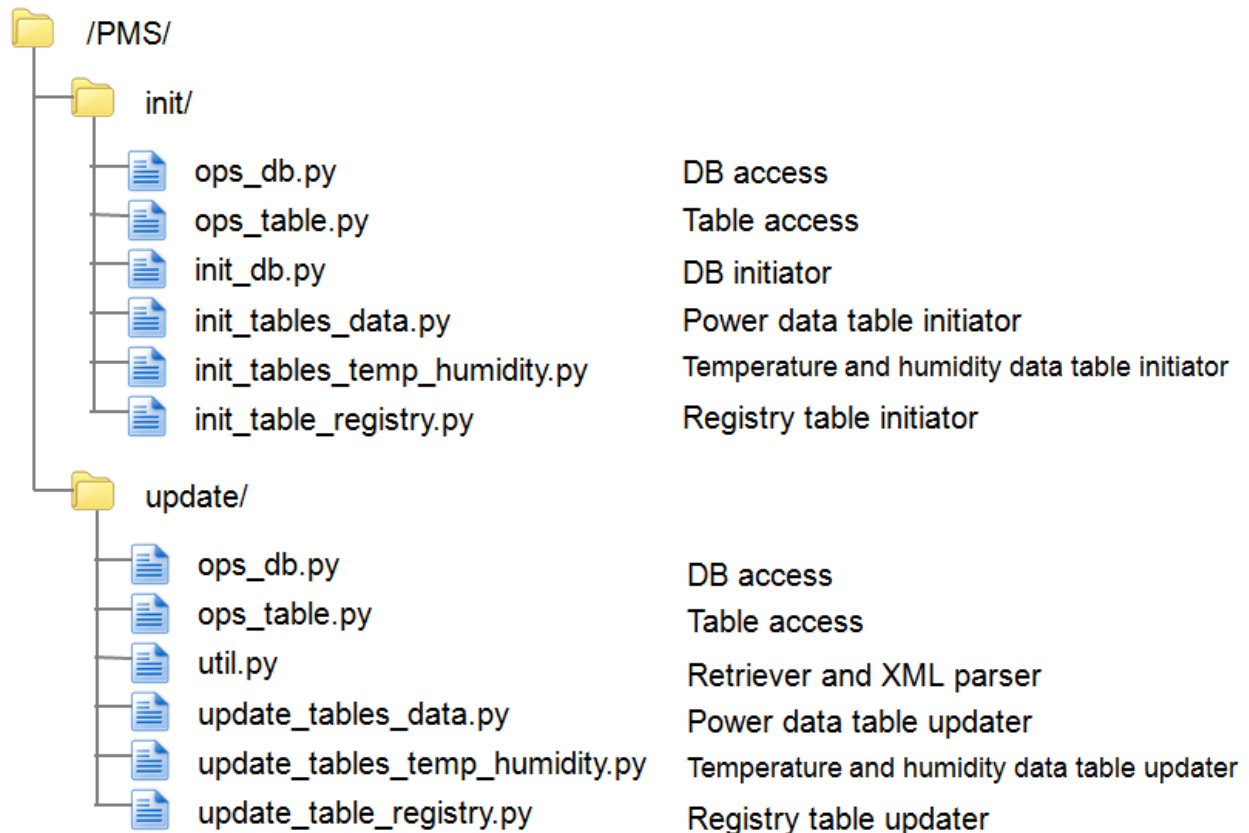


Fig.2.11. File structure of initiation module and update module

Name	Location	Parameters	Return value	Function
connect	ops_db.py	hostname, username, pswd	connection(obj)	connect to MySQL server
lst	ops_db.py	connection(obj)	dbn_lst(list)	list DB names
select	ops_db.py	connection(obj), dbn	N/A	select DB whose name is dbn
create	ops_db.py	connection(obj), dbn	N/A	create DB named dbn
drop	ops_db.py	connection(obj), dbn	N/A	drop DB named dbn
lst	ops_table.py	connection(obj), dbn	tbn_lst(list)	list table names in DB
create	ops_table.py	connection(obj), tbn	N/A	create table named tbn

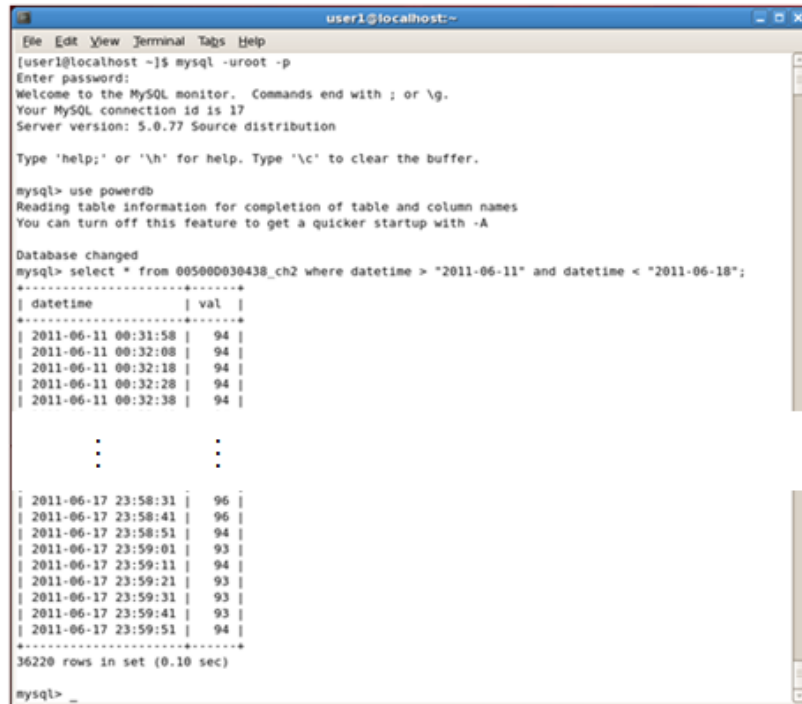
reg_create	ops_table.py	connection(obj), tbn	N/A	create registry table named tbn
drop	ops_table.py	connection(obj), tbn	N/A	drop table named tbn in DB
read	ops_table.py	connection(obj), tbn, st, et	val_lst(list)	read records from start time st to end time et
write	ops_table.py	connection(obj), tbn, datetime, val(int)	N/A	insert record with datetime and value, overwrite if record exists
csv2lst	init_tables _data.py	fn	lst(list)	convert csv file of name fn's content to mac address list
lst2chlst	init_tables _data.py	lstn	chlst(list)	convert mac address list to table names
csv2lst	init_tables _temp_ humidity.py	fn	lst(list)	convert csv file of name fn's content to mac address list
lst2chlst	init_tables _temp_ humidity.py	lstn	chlst(list)	convert mac address list to table names
datespan	util.py	st, et, delta(int)	datetime _lst(list)	make list of datetime strings between st and et
file_lst	util.py	st, et, url_dir	fn_lst	get file name list between datetime st and et
retrieve	util.py	url	file_lines(list) /0	get contents of remote file into a list, or 0 if not successful
xml_parser	util.py	xml_str	xml_dict(dict)	parse xml string into a dictionary
ch	inquiry.py	connection(obj), tbn,st,et	val_lst(list)	return sorted value list within table tbn , where the datetime is in between st and et
sort	inquiry.py	lst(list)	sorted_lst(list)	sort list into a increasing order of the first value
str2time	inquiry.py	str	datetime(obj)	convert string to

				datetime object
scale_ generate	inquiry.py	st, et, interval(int)	scale(list)	generate scale between st and et, with an interval
merge	inquiry.py	scale(list), data(list)	data_lst(list)	sort data according to the scale
power_sum	inquiry.py	lst(list)	sum(list)	get energy list with power list
view	inquiry.py	connection(obj), tbn,st,et, interval(int), raw	val_list(list) /sum(list)	get power data list if raw flag is “true”, energy data list if raw flag is “false”

Table.2.3. Detailed description of functions, parameters are strings if not specified, N/A in return value column means no return value

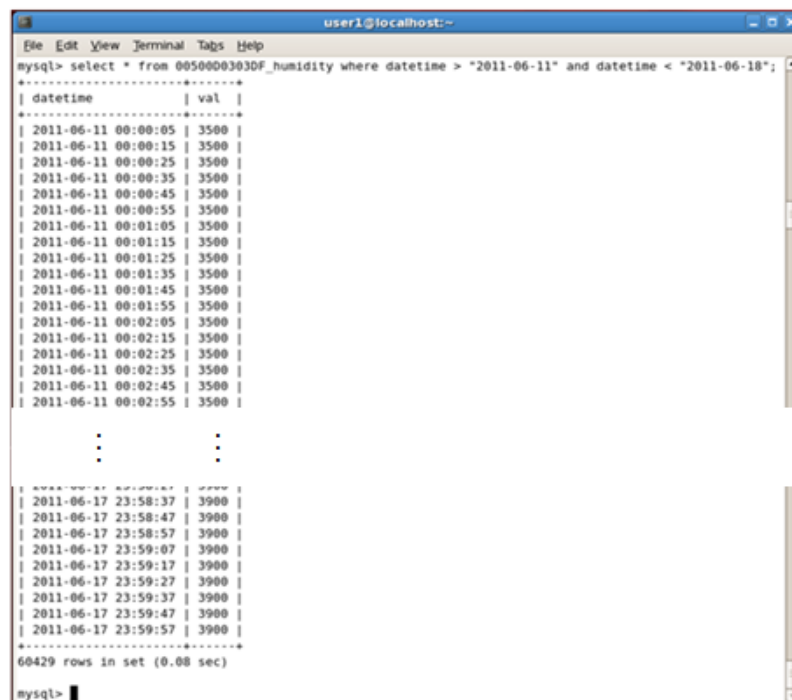
2.1.4. Result

The result of local access to DBMS is shown in figure 2.12-2.15. The result of intranet access to PMS is shown in figure 2.16-2.19. The result of local visualizer is shown in figure 2.20, 2.21.



```
user1@localhost:~  
[user1@localhost ~]$ mysql -uroot -p  
Enter password:  
Welcome to the MySQL monitor.  Commands end with ; or \g.  
Your MySQL connection id is 17  
Server version: 5.0.77 Source distribution  
  
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.  
  
mysql> use powerdb  
Reading table information for completion of table and column names  
You can turn off this feature to get a quicker startup with -A  
  
Database changed  
mysql> select * from 005000030438_ch2 where datetime > "2011-06-11" and datetime < "2011-06-18";  
+-----+-----+  
| datetime | val |  
+-----+-----+  
| 2011-06-11 00:31:58 | 94 |  
| 2011-06-11 00:32:08 | 94 |  
| 2011-06-11 00:32:18 | 94 |  
| 2011-06-11 00:32:28 | 94 |  
| 2011-06-11 00:32:38 | 94 |  
| : | : |  
| 2011-06-17 23:58:31 | 96 |  
| 2011-06-17 23:58:41 | 96 |  
| 2011-06-17 23:58:51 | 94 |  
| 2011-06-17 23:59:01 | 93 |  
| 2011-06-17 23:59:11 | 94 |  
| 2011-06-17 23:59:21 | 93 |  
| 2011-06-17 23:59:31 | 93 |  
| 2011-06-17 23:59:41 | 93 |  
| 2011-06-17 23:59:51 | 94 |  
+-----+-----+  
36220 rows in set (0.10 sec)  
  
mysql>
```

Fig.2.12. Data records of an appliance for one week in DBMS, view from local access



```
user1@localhost:~  
mysql> select * from 00500003030F_humidity where datetime > "2011-06-11" and datetime < "2011-06-18";  
+-----+-----+  
| datetime | val |  
+-----+-----+  
| 2011-06-11 00:00:05 | 3500 |  
| 2011-06-11 00:00:15 | 3500 |  
| 2011-06-11 00:00:25 | 3500 |  
| 2011-06-11 00:00:35 | 3500 |  
| 2011-06-11 00:00:45 | 3500 |  
| 2011-06-11 00:00:55 | 3500 |  
| 2011-06-11 00:01:05 | 3500 |  
| 2011-06-11 00:01:15 | 3500 |  
| 2011-06-11 00:01:25 | 3500 |  
| 2011-06-11 00:01:35 | 3500 |  
| 2011-06-11 00:01:45 | 3500 |  
| 2011-06-11 00:01:55 | 3500 |  
| 2011-06-11 00:02:05 | 3500 |  
| 2011-06-11 00:02:15 | 3500 |  
| 2011-06-11 00:02:25 | 3500 |  
| 2011-06-11 00:02:35 | 3500 |  
| 2011-06-11 00:02:45 | 3500 |  
| 2011-06-11 00:02:55 | 3500 |  
| : | : |  
| 2011-06-17 23:58:37 | 3900 |  
| 2011-06-17 23:58:47 | 3900 |  
| 2011-06-17 23:58:57 | 3900 |  
| 2011-06-17 23:59:07 | 3900 |  
| 2011-06-17 23:59:17 | 3900 |  
| 2011-06-17 23:59:27 | 3900 |  
| 2011-06-17 23:59:37 | 3900 |  
| 2011-06-17 23:59:47 | 3900 |  
| 2011-06-17 23:59:57 | 3900 |  
+-----+-----+  
60429 rows in set (0.08 sec)  
  
mysql>
```

Fig.2.13. Humidity data from a meter for one week in DBMS, the integer value is 100 times of actual value, for integer operation is much efficient than floats.

```

mysql> select * from 00500003030F_temp where datetime > "2011-06-11" and datetime < "2011-06-18";
+-----+-----+
| datetime | val |
+-----+-----+
| 2011-06-11 00:00:05 | 2700 |
| 2011-06-11 00:00:15 | 2700 |
| 2011-06-11 00:00:25 | 2700 |
| 2011-06-11 00:00:35 | 2700 |
| 2011-06-11 00:00:45 | 2700 |
| 2011-06-11 00:00:55 | 2700 |
| 2011-06-11 00:01:05 | 2700 |
| 2011-06-11 00:01:15 | 2700 |
| 2011-06-11 00:01:25 | 2700 |
| 2011-06-11 00:01:35 | 2700 |
| 2011-06-11 00:01:45 | 2700 |
| 2011-06-11 00:01:55 | 2700 |
| 2011-06-11 00:02:05 | 2700 |
| 2011-06-11 00:02:15 | 2700 |
| 2011-06-11 00:02:25 | 2700 |
| 2011-06-11 00:02:35 | 2700 |
| 2011-06-11 00:02:45 | 2700 |
| 2011-06-11 00:02:55 | 2700 |
| ... | ... |
| 2011-06-17 23:58:37 | 2640 |
| 2011-06-17 23:58:47 | 2640 |
| 2011-06-17 23:58:57 | 2640 |
| 2011-06-17 23:59:07 | 2640 |
| 2011-06-17 23:59:17 | 2640 |
| 2011-06-17 23:59:27 | 2640 |
| 2011-06-17 23:59:37 | 2640 |
| 2011-06-17 23:59:47 | 2640 |
| 2011-06-17 23:59:57 | 2640 |
+-----+-----+
60429 rows in set (0.09 sec)

mysql>

```

Fig.2.14. Temperature data from a meter for one week in DBMS, the integer value is used for the same consideration as in humidity table.

```

mysql> select * from owner_app_ch_loc;
+-----+-----+-----+-----+
| owner | app | ch | location |
+-----+-----+-----+-----+
| Okamoto-sensei | fax. | 005000030451_ch1 | Okamoto-sensei.home |
| Okamoto-sensei | phone. | 005000030451_ch2 | Okamoto-sensei.home |
| Okamoto-sensei | router.FTTH | 005000030451_ch3 | Okamoto-sensei.home |
| Okamoto-sensei | router.wireless | 005000030451_ch4 | Okamoto-sensei.home |
| Okamoto-sensei | TV_recorder. | 005000030452_ch1 | Okamoto-sensei.home |
| Okamoto-sensei | TV. | 005000030452_ch2 | Okamoto-sensei.home |
| Okamoto-sensei | printer. | 005000030452_ch3 | Okamoto-sensei.home |
| ... | ... | ... | ... |
| shared | dehumidifier. | 00500003030F_ch1 | Hongo.130 |
| shared | dehumidifier. | 0050000303E0_ch1 | Hongo.128 |
| shared | router.wireless | 0050000303E0_ch2 | Hongo.128 |
| shared | kettle. | 005000030438_ch1 | Env.223 |
| shared | fridge. | 005000030438_ch2 | Env.223 |
| shared | printer. | 005000030438_ch3 | Env.223 |
| Meng | display.2 | 005000030424_ch1 | Env.223 |
| Meng | desktop. | 005000030424_ch2 | Env.223 |
| Meng | display. | 005000030424_ch3 | Env.223 |
| Meng | router. | 005000030424_ch4 | Env.223 |
| shared | storage.VIS_TR4 | 005000030434_ch1 | Env.223 |
| shared | storage.VIS_TR3 | 005000030434_ch2 | Env.223 |
| shared | storage.VIS_TR2 | 005000030434_ch3 | Env.223 |
| shared | storage.VIS_TR1 | 005000030434_ch4 | Env.223 |
| Matsushita | display. | 00500003041F_ch1 | Env.223 |
| Matsushita | notebook. | 00500003041F_ch3 | Env.223 |
| shared | router. | 005000030435_ch1 | Env.223 |
| Yanrong | desk_lamp. | 005000030435_ch2 | Env.223 |
| shared | router. | 005000030431_ch1 | Env.223 |
| Qiang | display. | 005000030431_ch2 | Env.223 |
| Qiang | notebook.dell | 005000030431_ch3 | Env.223 |
| Qiang | desktop. | 005000030431_ch4 | Env.223 |
| Qiang | notebook.thinkpad | 005000030421_ch3 | Env.223 |
+-----+-----+-----+-----+

```

Fig.2.15. Registry table in DBMS

The data records can be accessed within the intranet, the format of URL is (parameters are contained in "<>"):

```
http://<host_url>/cgi-bin/<filename>?
ch=<meter ID>_<channel>&
st=<YYYY>d<MM>d<DD>s<hh>c<mm>c<ss>&
et=<YYYY>d<MM>d<DD>s<hh>c<mm>c<ss>&
interval=<seconds>&raw=<true/false>
```

Filename can be:

1. power_single.py: return single/multiple power data records, channels are separated by "-"
2. humidity_single.py: return single humidity data records
3. humidity_multi.py: return multiple humidity data records, channels are separated by "-"
4. temp_single.py: return single temperature data records
5. temp_multi.py: return multiple temperature data records, channels are separated by "-"

Meter ID and the channel must be consistent with file type, for example, value and/or unit mismatch will happen when intend to get power data from a humidity channel. When in the energy mode, data displayed on the browser are summed with interval parameter, measured in seconds. The raw flag must be set to false. The following are examples:

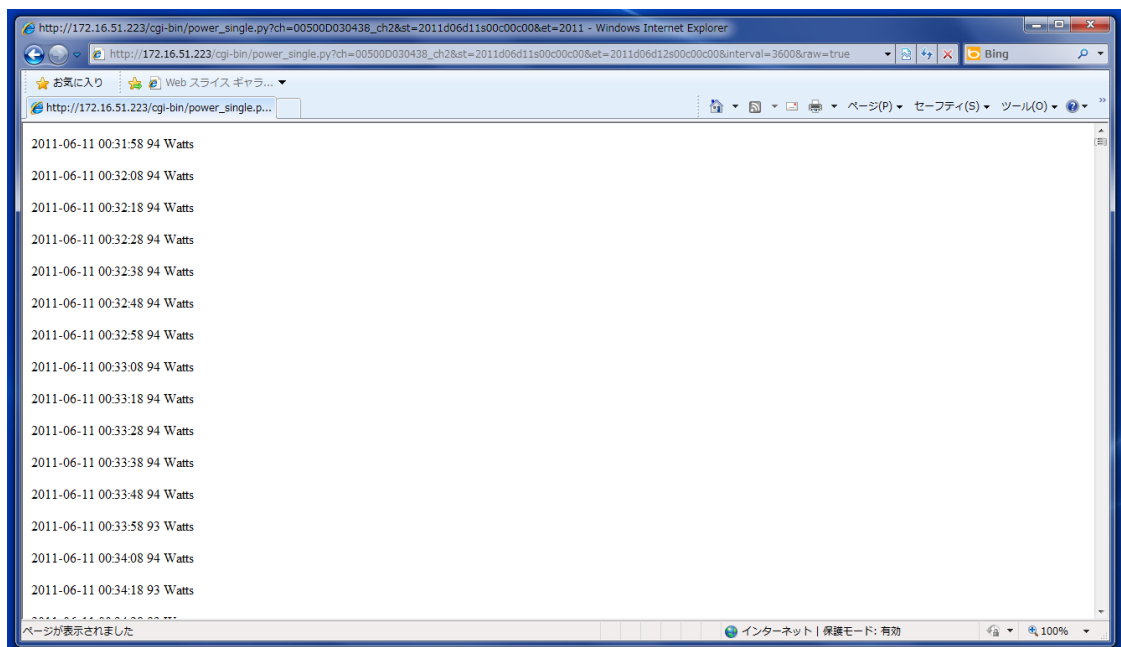


Fig.2.16 Power data view for a single appliance on browser within intranet, the unit of power data is Watt

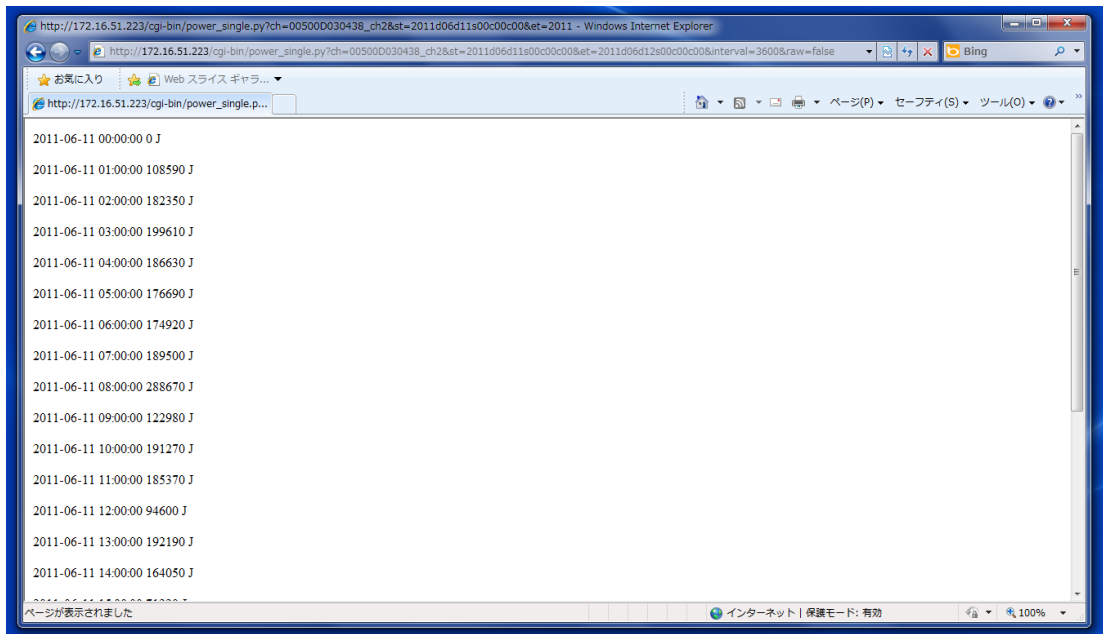


Fig.2.17. Energy data view of a single appliance with 1 hour's interval, the unit is J, the first zero line has no meaning

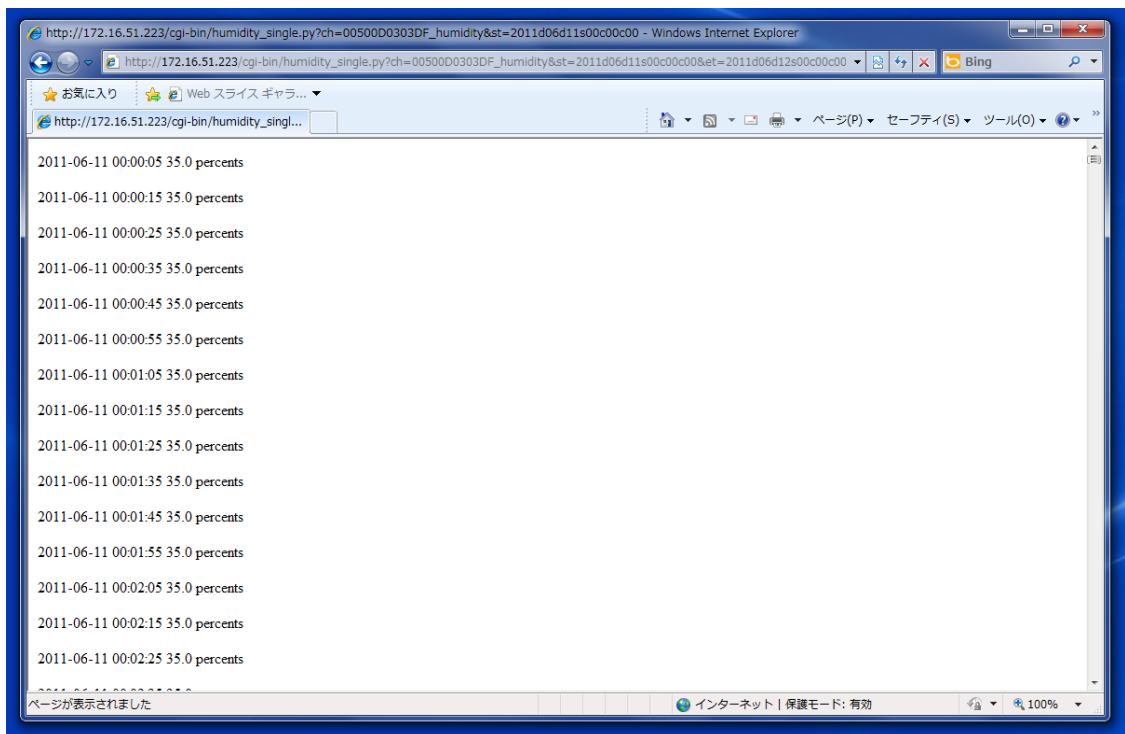


Fig.2.18. Humidity data view of a single meter, the unit is percent.

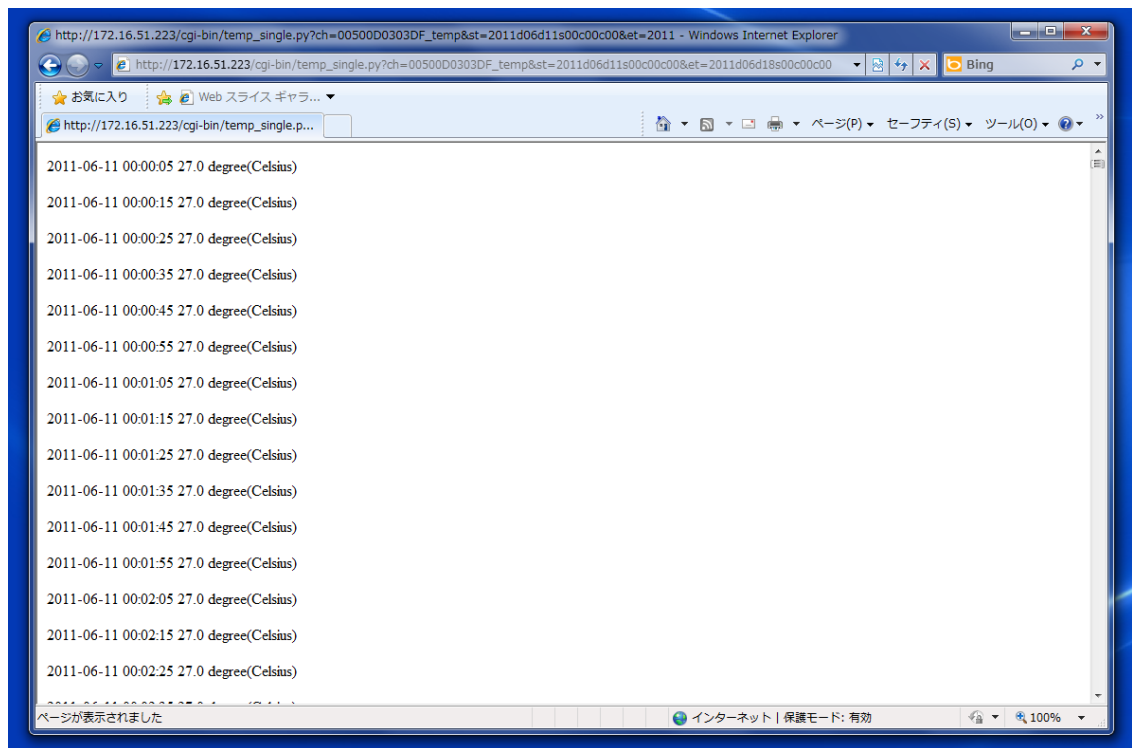


Fig.2.19 Temperature data view of a single meter, the unit is Celsius degress

When the power/energy data are copied in to plain text file, the local visualize can be used to demonstrate the data chart simply. The following are examples:

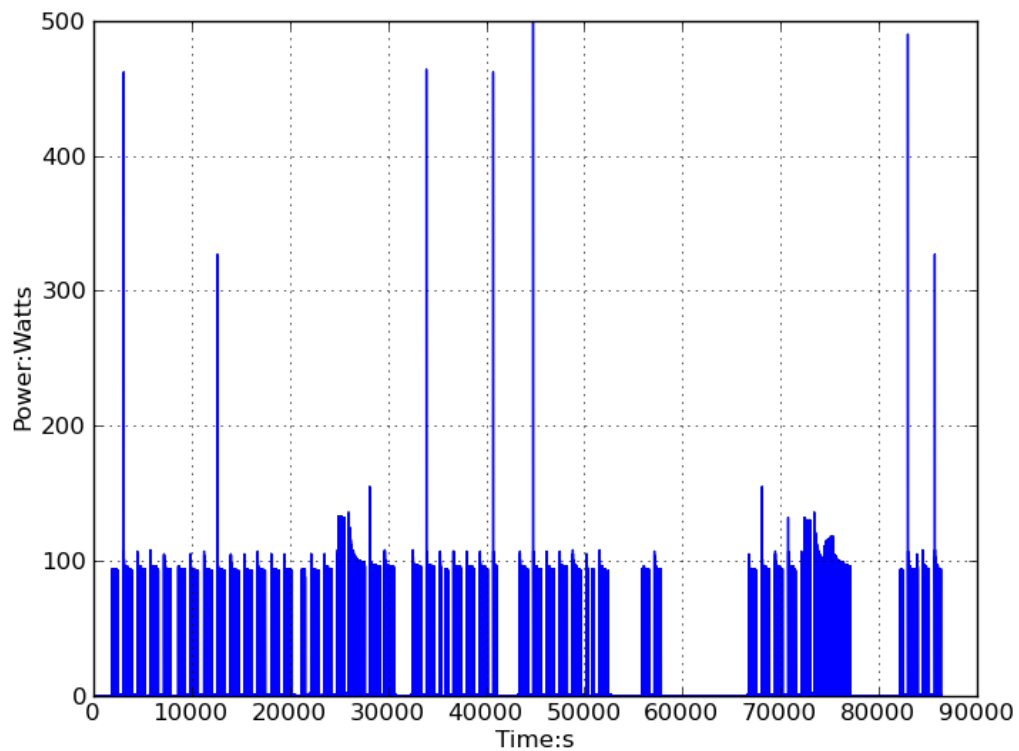


Fig.2.20. Power view of an appliance on local visualizer

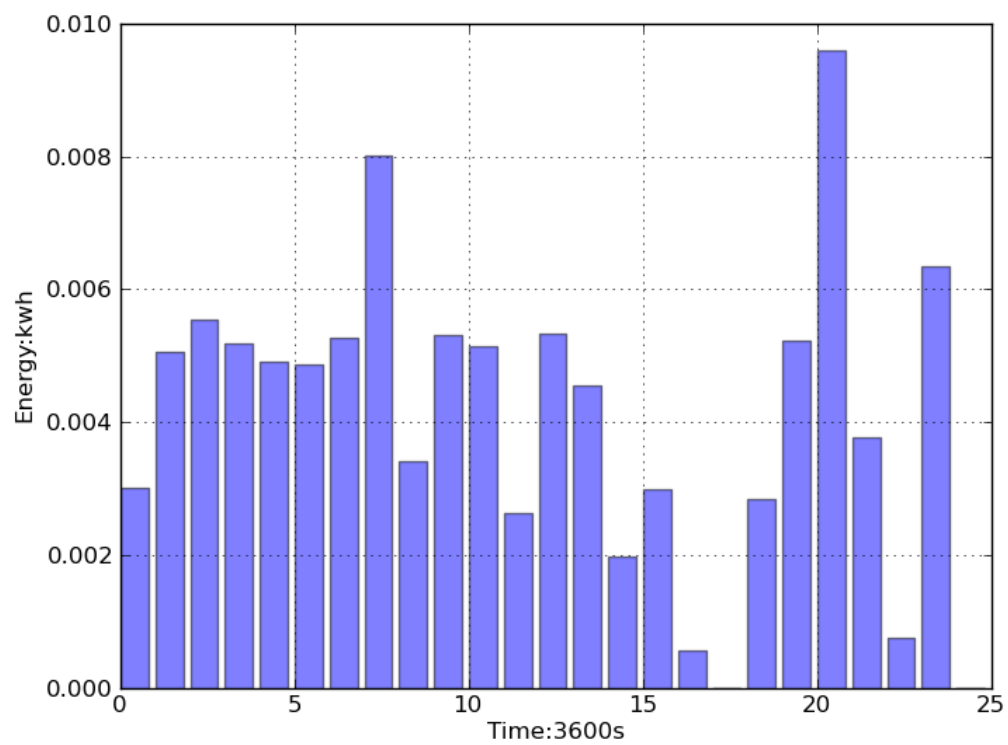


Fig.2.21. Energy view of an appliance on local visualize

2.2. Simulation environment

2.2.1. Overview

Considering a real power consuming environment, such as a community, is consist of cells. The cells are classified such as houses, apartments, offices, research rooms, stores... The class of a cell is decided by its main components: human users. Within each class, there can be a number of instances, representing power consuming units of the same type in practical. Figure 2.22 shows the composition example of a community.

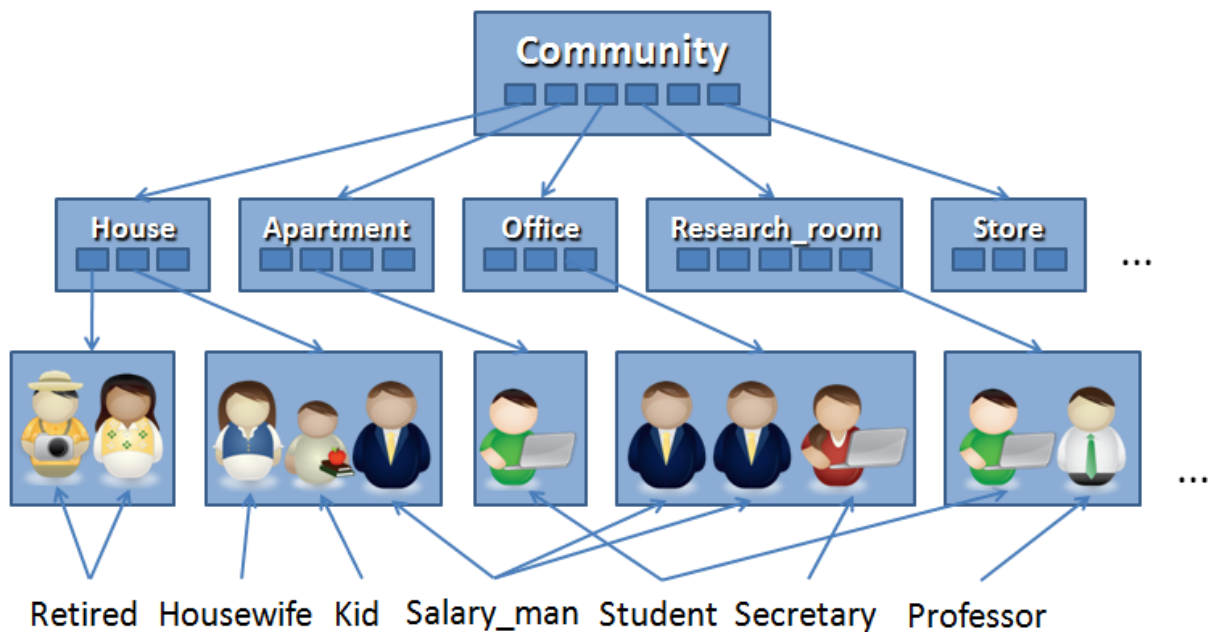


Fig.2.22. Compositions of cells in a community

The human users in cells can be classified according to their power consuming behaviors. This configuration is based on the observation that people's behaviors normally vary with factors such as work, age, gender of the person.

Within a cell of class research room, as shown in figure 2.23, for example, there are human users of class students and appliances of a variety of classes, such as lightening, desktop PCs, displays, fridges and so on. Human users are the indirect power consumers, whereas the appliances are actually the direct power consumers.

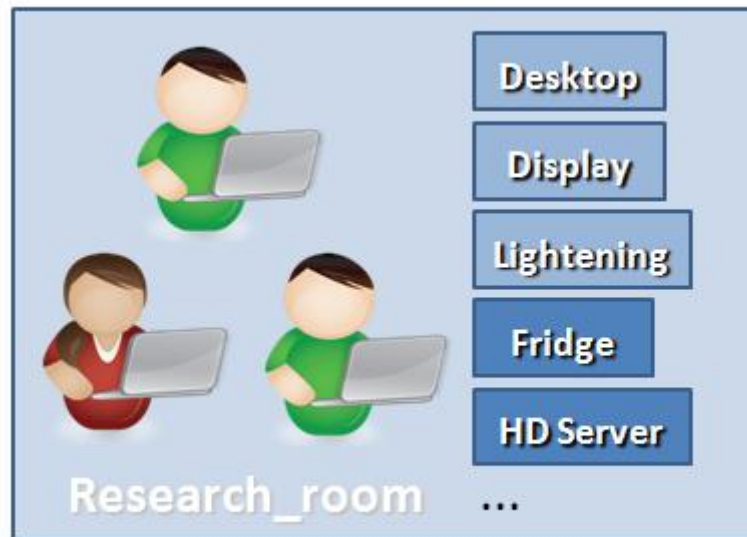


Fig.2.23. Cell composition example: research room

According to the observation above, in a simulated power consuming environment, the components are appliance simulators and human behavior agents. We can have flexibility in modifying the composition, in both the quantities and types of appliances. Instead of fixed power curves, we consider interactions between human agents and power appliances.

A multi-agent system is suitable for building such a kind of simulation environment.

2.2.2. Appliance simulators

2.2.2.1. State machine in modeling appliance

In power behaviors, appliances may be of several working phases, as shown in figure 2.24:

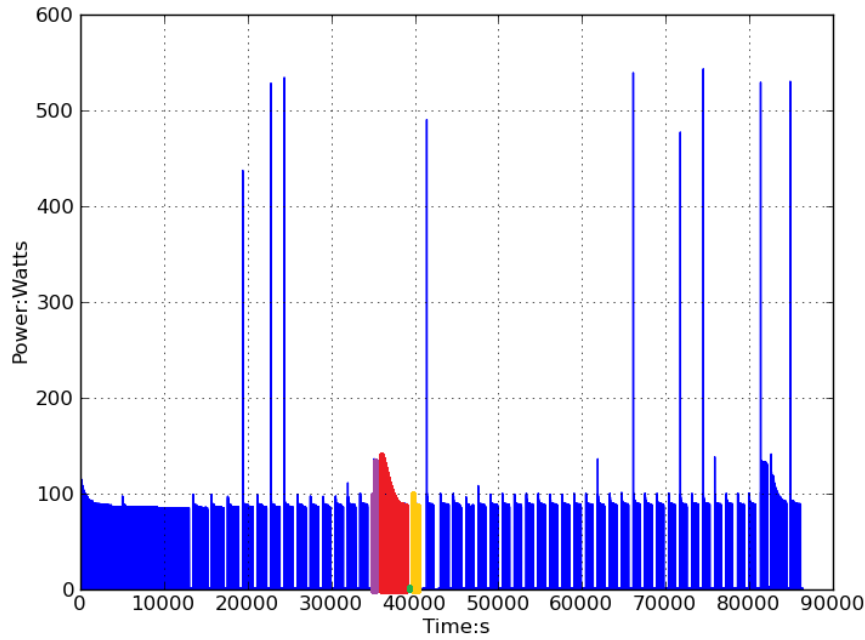


Fig 2.24. Working phases of a refrigerator, purple part: pre-high phase, red part: high power phase, green part: standby phase, yellow part: low power phase

Each phase has a start time and end time, and the transitions between working phases are decided by some factors such as internal timer or external access. Identical appliances on different working phase behave differently, which result in different appearances in power curves. The working phases here are defined according to the appliance's external power behaviors, for the consideration that, its detailed internal mechanism are invisible for ordinary users, so the modeling can be done conveniently.

In our approach, we model the appliance with state machines. A state machine possesses several states, representing working phases of a practical appliance. The transitions between working phases are modeled as a set of rules to transit between states in the state machine as shown in figure 2.25.

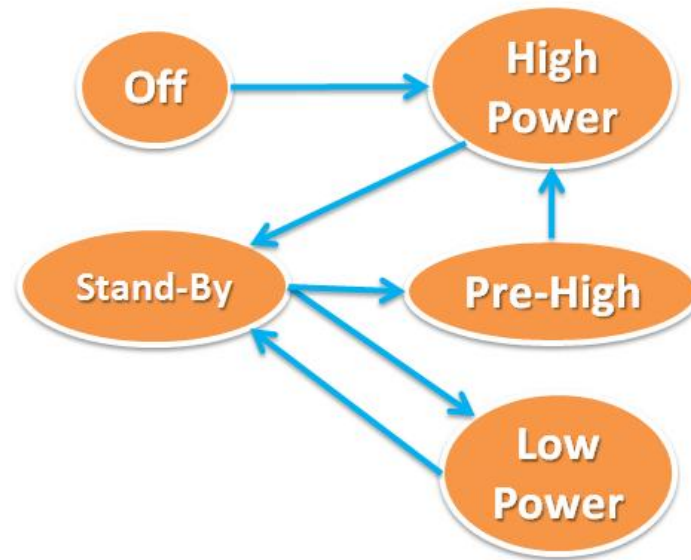


Fig.2.25 State machine modeling for refrigerator

2.2.2.2. Non-interactive and interactive appliances

Appliances can be classified into non-interactive and interactive, according to whether its transitions are responses to human user's requests. Figure 2.26 shows an example of comparison of non-interactive and interactive appliances' power curves.

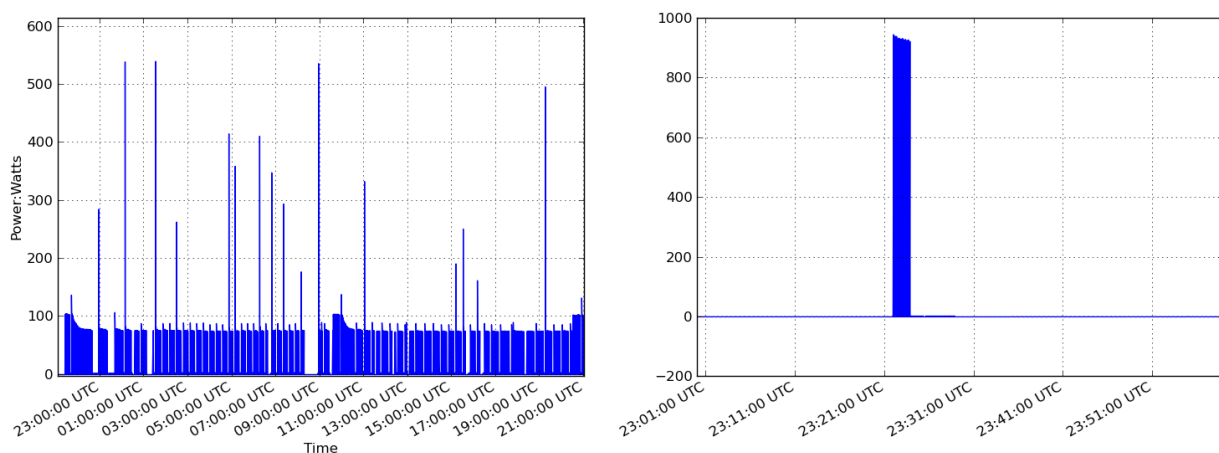


Fig 2.26 Comparison of non-interactive and interactive appliances' power curves

Non-interactive appliance's transitions happening or not depends only on its internal timer(s), whereas interactive appliances' transitions are caused by not only internal timer but also users' requests. Non-interactive type appliances include: fridge, kettle, phone, server, router and so on; Interactive type appliances includes: microwave oven, air conditioner, desktop PC, laptop PC, display, printer, TV and so on. Note that appliances such as phones and kettles, are of the non-interactive type, that does not mean they do not respond to user's request, however, on power consuming behaviors, phones are normally periodically charged, kettles are normally periodically powered, regardless of users' accessing.

2.2.2.3. Permissions on interactive appliances

As for interactive appliances, some appliances can only serve one user at one time, whereas some can serve multiple users at one time. Some appliances serve their owners only, whereas some serve a group of users. We further classify interactive appliances into non-mutual exclusive and mutual exclusive according to the number of users they serve at one time; shared and personal according to who the appliances serve. Figure 2.27 shows the design.

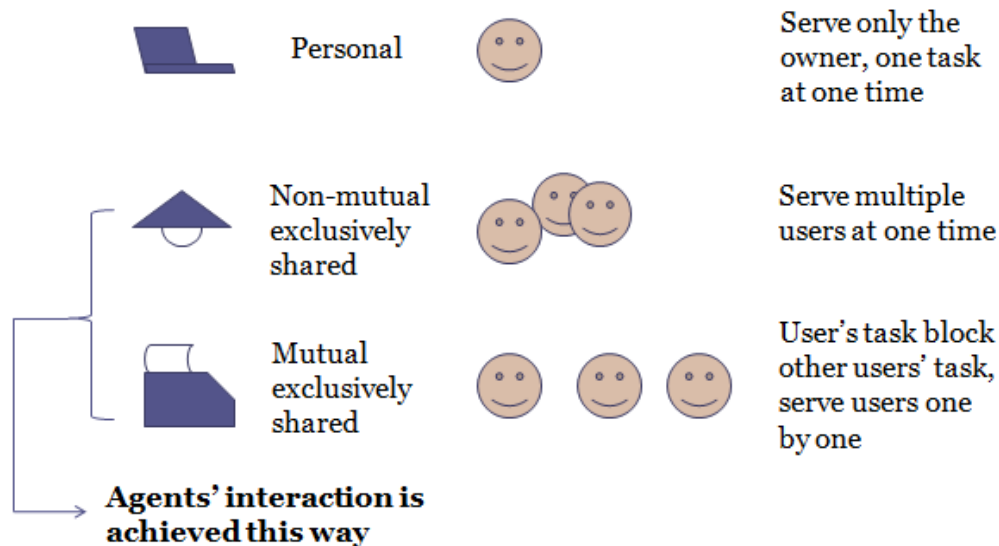


Fig.2.27 Personal/shared and non-mutual/mutual exclusive appliances

The interactions between human users are mainly presented in the competitive usage of shared appliances. When time conflicting usages happen to a shared mutual exclusive appliance, the tasks for users' requests are queued and executed one by one. When time conflicting usages happen to a shared non-mutual exclusive appliance, the working time usually extends to satisfy the longest time requests, as shown in figure 2.28.

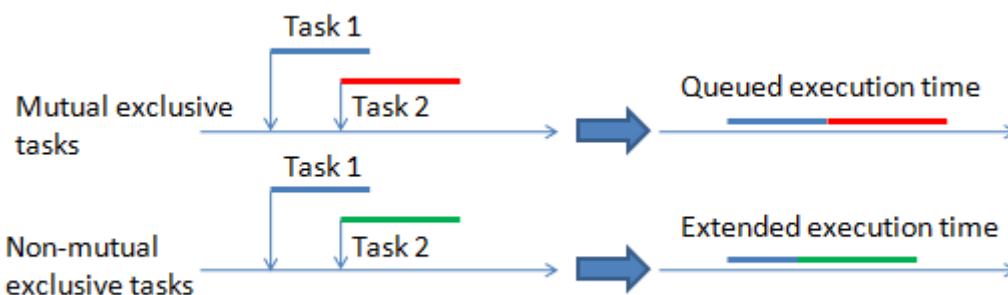


Fig.2.28 Mutex/non-mutex appliance's response to time conflicting requests

Examples of shared mutual exclusive appliances are: microwave oven, printer, etc. Examples of shared non-mutual exclusive appliances are lightening, air conditioning, etc.

2.2.2.4. Power consumption functions

Within each working phase, the power measures of appliance can be viewed as a power consumption function with the timer under current state and/or external timer (such as local time) as variables. The function is manually selected to make a base line best fitting the power curve, all we need to do is to find the parameters for the function. Figure below shows an example of using an exponential curve to fit a power curve.

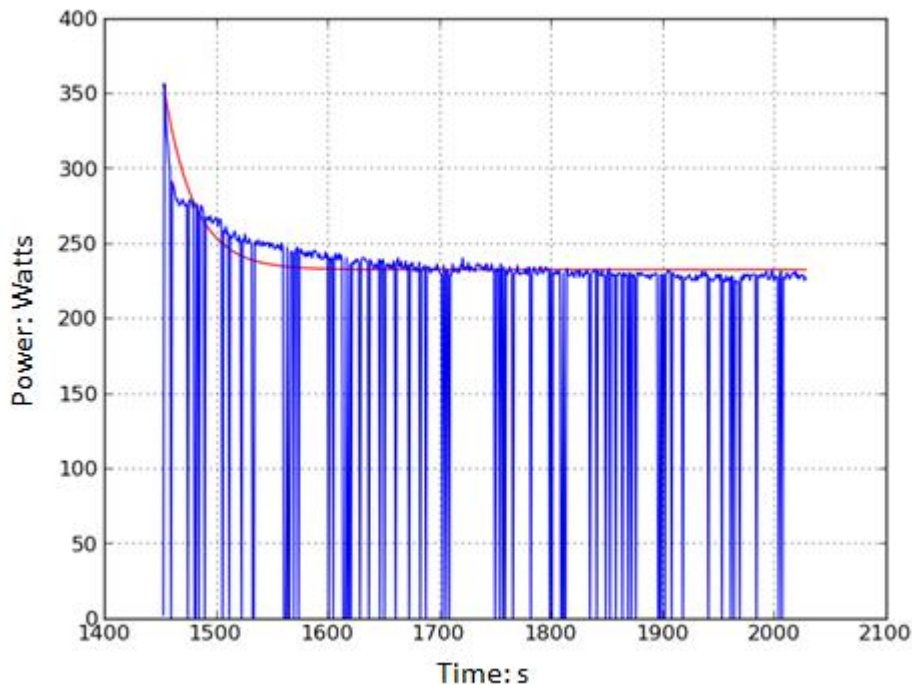


Fig.2.29. Power consumption function of time, red curve is the plot of output data, fitting the practical data in blue lines under a working phase

In practical data, the power curve is discrete as: the capturing of power measure is limited to several seconds in time resolution and data loss happens due to network instability sometime. The fitting function, however, generates a smooth base line that only approximately agrees with the practical one.

Usually the power value of an appliance distorts, and abnormal conditions (example shown in figure 2.30) happen in the curve; it is especially obvious in appliances of drastic power fluctuations. In modeling the usage into the power consumption function, they are treated as distortions and abnormal conditions of the fitting curve.

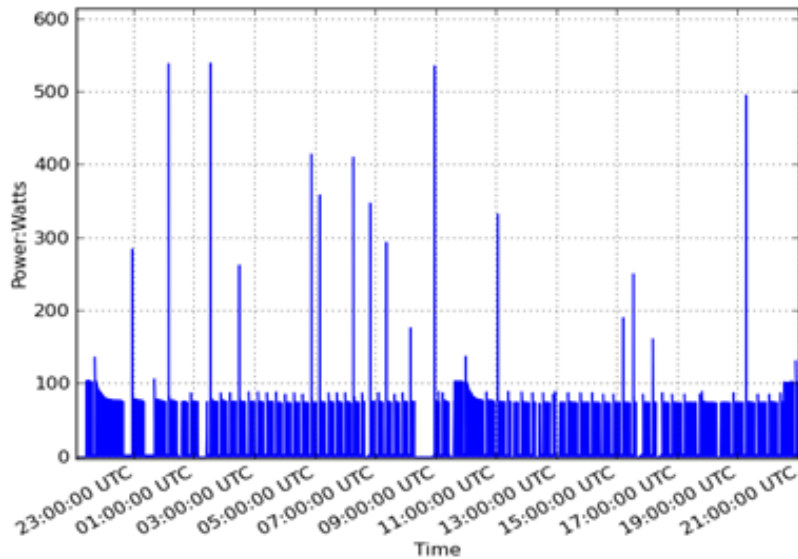


Fig.2.30. Abnormal conditions in long term captured practical data

2.2.2.5. Control logic of appliance simulator

To act as an appliance of specific class, the simulator reads profiles, which contains the information about working phases and transitions, time and power parameters of each phase. The work flow of an appliance simulator is described as the following:

1. Initiating appliance simulator, read parameters from profile.
2. Check if there is a pending request from agent in request buffer, if yes go to step 8, if no, go to step 3.
3. Check transition chart, under current state, if the timer exceeds the wait limit, if yes go to step 7, if no, go to step 4.
4. Output power measure using current power consumption function and current timer.
5. Timer increases.
6. Loop to step 2.
7. Reset timer under current state,
Find next applicable state in transition chart,
Set power consumption function of that state the current power consumption function,
Switch to the state found, go to step 5
8. Pop the request out from the request buffer as the current request.
9. Check route table with current request, under current state, if the timer exceeds the wait limit, if yes go to step 10, if no, go to step 4.
10. Reset timer under current state,
Find next applicable state in route table,
Set power consumption function of that state the current power consumption function,
Switch to the state found, go to step 5

The request may come to appliance anytime, a request buffer inside appliance is used to store incoming requests, and may later allow the simulator perform task scheduling. When the appliance simulator run in a time frame, it checks the request in the buffer, and decides whether to respond to the request.

A detailed function list in the appliance simulator is in the table 2.4:

Name	Parameter	Return Value	Function
config_file_read	profile_filename	params_d (dictionary)	Call all applicable parameter reading functions
node_table_read	param_str	node_table (dictionary)	Read node table string into a dictionary
transition_chart_read	param_str	transition_chart (dictionary)	Read transition chart string into a dictionary
route_table_read	param_str	route_table (dictionary)	Read route table string into a dictionary
pcf_params_read	param_str	pcf_params (dictionary)	Read power consumption function parameter string into a dictionary
distortor	base_val (int), distortion_rate (float)	distorted_val (int)	Add distortion to power measure value
pcf	pcf_params, current_timer (int), global_timer (int)	pwr_val (int)	Get current power measure value, distortions not included
__init__ (in class function)	profile_filename, init_state	N/A	Initiate instance for appliance simulator class
run (in class function)	N/A	N/A	Output power measure value for a time point
rcv (in class function)	request_str	N/A	Receive user request (only in interactive appliance)
get_request (in class function)	N/A	request_str/ "n/a"	Read next request from request buffer
new_day (in class function)	N/A	N/A	Reset global timer

Table 2.4. Functions in appliance simulator

2.2.2.6. Parameters in profiles

The appliance profile contains the following information:

1. Transitions, it is usually decided by the type, the internal functioning mechanism of the appliance.
2. Time parameters in the transitions, including minimum wait limit for each state, and periods for periodically appearing states.
3. The power consumption function type and parameters at each state, this information instructs the simulator how to give out the power value specifically
4. A route table is in the profile of a interactive appliance, it resembles the transition table but with a different meaning, it instructs the simulator how to transit when a request incomes.

The format of the profile is demonstrated by the following example of an air conditioner's (interactive appliance) profile:

```
node_table=off:0,0;
            w1:12000,0.2;
            w2:0,0;
            working:0,0

transition_chart=off,w1:n/a;
                w1,off:0,0;
                w1,w2:0,0;
                w2,working:0,0;
                working,w1:0,0

route_table=working,off,w1:0,0;
            working,w1,w2:2500,0.1;
            working,w2,working:1000,0

pcf_params=off:0 | 0/0/+ | 0,0,0;
            w1:0 | 10/0.2,0.01/+ | 0,0,0;
            w2:0 | 500/0.2,0.03/+ | 0,0,0;
            working:2 | 260,90,46000,10000/0.05,0.01/+ | 0,0,0
```

Node table contains the information of states, with the state name and minimum wait time parameters, the parameters are the mean, and variance of the wait time, which will be decided randomly according to these parameters in runtime. Transition chart specifies the switching among states, the names before the colon are the current state and the next state names, the parameters after the colon are the mean and variance of the period of the transition, which will be decided randomly according to these parameters in runtime. The “n/a” in node table and transition chart means the unlimited wait time, in another word, the state would never end/ the transition would never happen, this design is used for interactive appliances, that only when a incoming request can trigger a state change (as transiting according to the route table).

In the route table, parameters before the colon are the desired working state

name, current state name, the next state name; parameters after the colon are the mean and variance parameters of wait limit. “n/a” in route table means “wait for the current state end”.

Power consumption function parameters for each state are separated by “/” into 3 parts accordingly: base function parameters, distortion parameters, abnormal condition parameters. In the first part of base function parameters, the number before “|” specifies the type of function, and the numbers after it specifies the parameters for the function. There are 3 types of function currently:

1. $pcf(t) = p$, type 0, t is timer under current state
2. $pcf(t) = p_1 + p_2 \cdot e^{p_3 \cdot t}$, type 1, t is timer under current state
3. $pcf(t) = p_1 + p_2 \cdot e^{-\frac{(t-p_3)^2}{p_4^2}}$, type 2, t is a global timer

The distortion part specifies the rate of distortion. In the abnormal condition part, the symbol before “|” specifies the condition of abnormal values, “+” for only higher values are made to the final output, “-” for the lower, “?” for all. The parameters after “|” are the base value, distortion rate, happening probability of the abnormal contion.

2.2.2.7. Profiling of appliances

To allow more flexibility for future expansion, the data is separated apart from program. The appliance simulators use profiles made according to practical data as the basis to generate simulation data. The typical procedure of making such a profile is shown in figure 2.31.

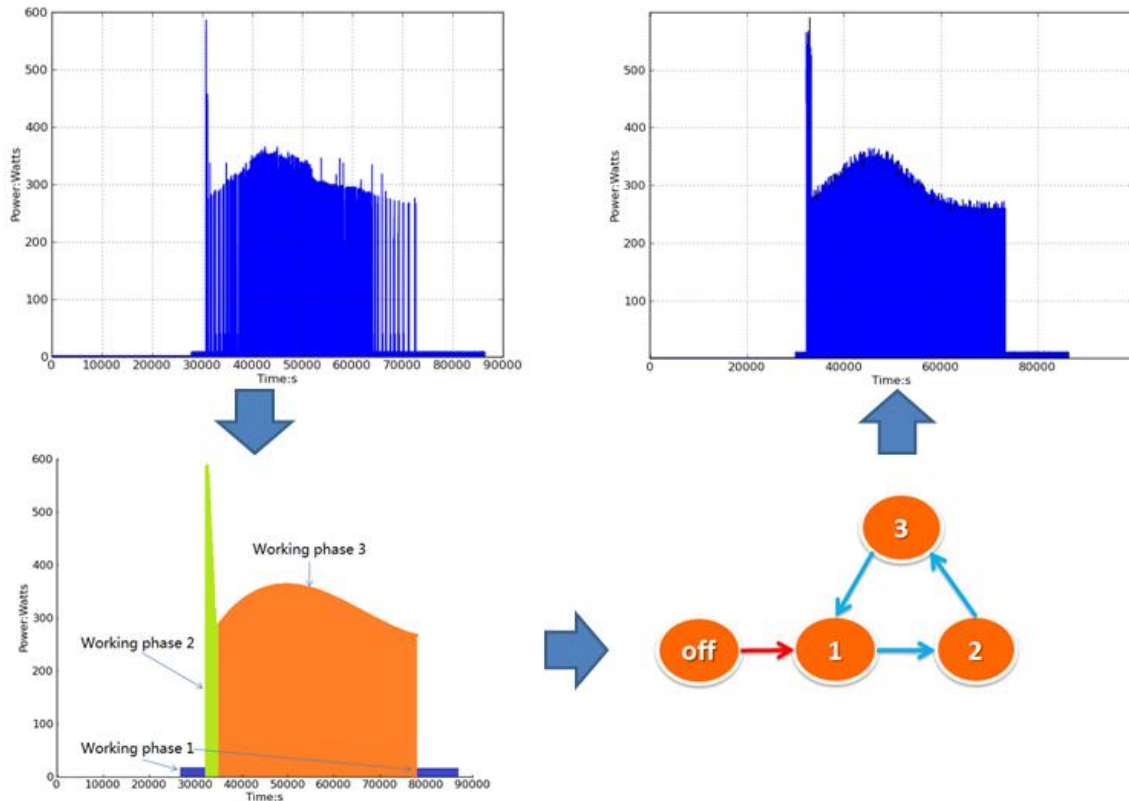


Fig.2.31 Procedure of making a profile for an appliance: Get the practical power curve of the appliance (upper left), abstract to working phases (lower left), decide state transitions and set parameters (lower right), compare the simulation result and modify the parameters (upper right)

The first step is to abstract working phases, according to the external power behaviors. Then determine the transitions of states for the state machine in modeling. Every node has a minimum wait limit: the internal timer can trigger transitions only when exceeds the wait limit. For nodes branching in directions to next states, there is a period timer for each transition, transition with a longer period is of a higher priority to be conducted. After determining the transitions and parameters for each node and for branching transitions, we need to decide the type and parameters for the power consumption function of each state. The type is quite arbitrarily decided, in most cases the base line of the power consumption is just a simple linear line. In refrigerator's working phases, an exponentially decreasing style line form is used as the base line. The distortion and abnormal conditions are also included in the parameters; they are usually given with certain randomness.

For interactive appliances, the transitions happen not only on when the timer

triggering them, but also on incoming of user's requests. When responding a request, the transitions of an appliance can be different from that in transiting without requests. For example, appliance may be turned on to some state that it would never go to itself. Based on this consideration, a routing table is made for interactive appliances, describing their transitions when requests present. The routing contains the wait limits for each state, without branching conditions (the destination of next node must always be decided).

As appliances of the same class usually behave similarly in working phases (if not, make a new profile as a new class), the structural parameters are the same among all the instances of the class, difference in appearances of models (within same class) are presented by the difference in numerical parameters such as time lengths and power consumption function parameters. The numerical parameters on runtime are decided by the parameters in profile with randomness. The randomness is also used in making different appearances of the power curve for identical appliance.

2.2.3. Agents

Human agents in simulation environment play the role of request generators, their requests are the causal factor of usage from interactive appliances. They make the simulation of cells more realistic, with different classes and numbers of human agent contained, the cells can be different in appearances.

2.2.3.1. Layer structure in modeling user behavior

Human users can be modeled as a set of behaviors. At different time of a day, human user performs approximately the same behavior. For example, people usually start working in the morning and having lunch around noon. What types of behaviors a human perform, from when and for how long the behavior is performed is decided by the class of human user. Human users of the same class perform similar behavior with close time parameter.

For individual users, some behavior happens only in another behavior's performing. For example, in an office, usage with PC usually happens when working is started, usage of printer always happen during the usage of PC.

Base on this observation, in agent, we model the behaviors as layered structure of procedures, each procedure represent a behavior of human user, as shown in figure 2.32.

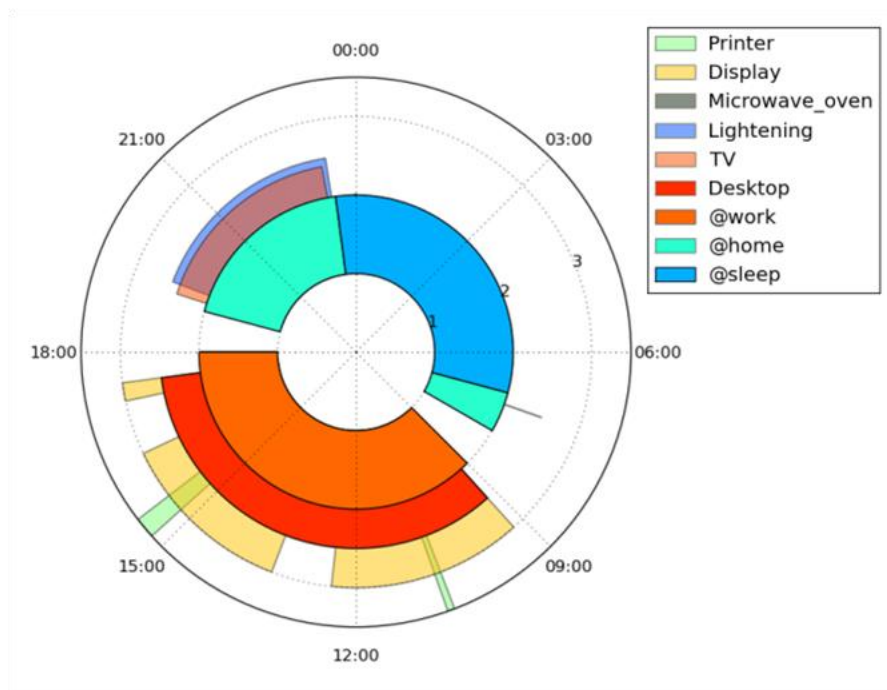


Fig.2.32 Example of procedures of a student type agent in a day

Human user's behaviors are recorded into a list in surveying. The dependence and temporal parameters are decided manually for each procedure in the list. There are

two types of procedures, grouped according to: whether it involves usage of certain appliances. The procedure that does not involve usage of appliances, which means no power usage, is called status. The procedure that does not involve usage of appliances, which means no power usage, is called activity. Though status are not power consuming, it provides a dependency for other statuses and activities. For example, at work is a status that does not consume power directly, but it is the parent procedure for the usage of PC and grand-parent for the usage of printer.

2.2.3.2. Randomness in human behavior

Exact time parameters of individual human users' behaviors seem to be random and are difficult to predict, but in case of a large number of agent, certain behavior's parameters is normally distributed. With this statistic assumption, time parameters of behaviors of a group of human users within a same class should be predictable. Agent simulators use the parameters to generate appearances of human user within certain class.

We also noticed that even the identical person, may perform the same behavior at different time day to day. As shown in figure 2.33, wake up time of a human during a year.

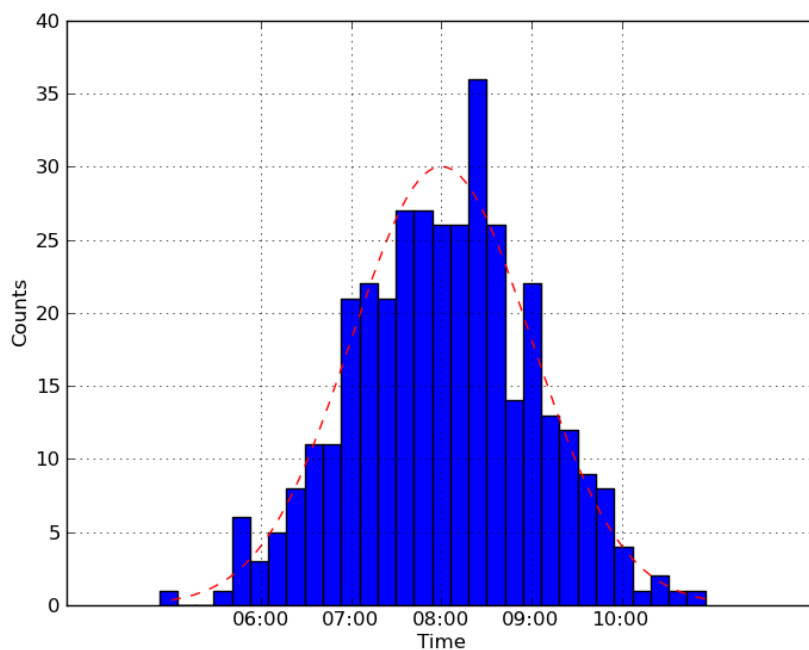


Fig. 2.33. Example of a human's wakeup time during a year, with mean value of 8:00 and variance of 1 hour

Similarly to the case of different persons, in case of the identical person, the distribution of certain time parameters accords with normal distribution. Thus the agent simulator may generate request at different times during a day according to the type of human user it emulates and the randomness in runtime.

2.2.3.3. Control logic of agent simulator

Similarly as appliance simulator, agent simulator read profiles of human users to present different behaviors of individual users. Users of the same class usually possesses similar behaviors (if not, a new class of users should be made), the procedures in the agent profiles are the same among all the instances of the class, difference in appearances of individuals (within same class) are presented by the difference in probability and/or the time parameters of these processes. The parameters on runtime are decided by the parameters in profile with randomness. The randomness is also used in making different appearances of identical agent in different days. The work flow of agent simulator is described as the following:

1. Initiating agent simulator, read parameters from profile.
2. Check current procedure in procedure list, if the procedure satisfies transition conditions, go to step 3, if not, go to step 4.
3. Perform transit actions for current procedure
4. Set the next procedure as current procedure, loop to step 2.

Each procedure has two status: running and waiting, on each time frame, agent checks every procedure in its procedure list, and set procedures satisfied transition conditions into transited status after conducting transit actions, as shown in figure 2.34:

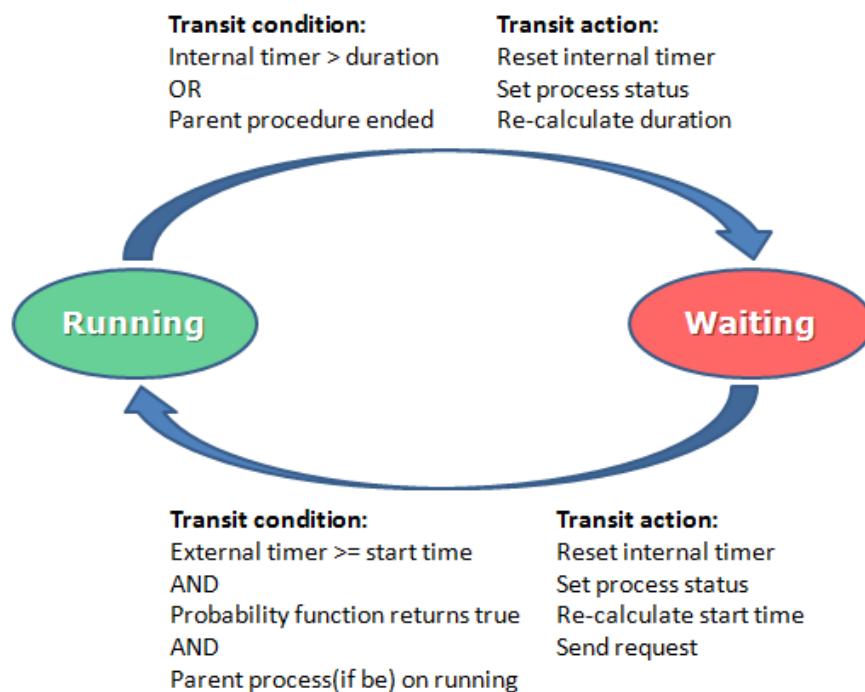


Fig.2.34. Control logic of each procedure in agent

A detailed function list is in the table 2.5:

Name	Parameters	Return values	Function
nd	mu(int), sigma(float)	X(int)	return random variable X with

			normal distribution specified by mu and sigma
read_profile	profile_name	params(dictionary)	read profile and return a dictionary of parameters
is_blocked	group, proc(dictionary)	True/False	check whether the procedure is blocked by another procedure in the same group
in_prob	p(float)	True/False	check whether a it is in the probability specified by p
__init__ (in class function)	profile_name	N/A	initiate an agent class instance using profile named profile_name
act (in class function)	N/A	N/A	behave in every time frame
send_req (in class function)	proc_key	N/A	send request for procedure of which the key is proc_key

Table 2.5. Functions in agent simulator

2.2.3.4. Parameters in profiles

For each procedure possesses parameters such as:

1. Procedure ID
2. Parent procedure ID
3. Type, status/activity
4. Start time/start probability
5. End time/duration
6. Target appliance (Activity procedures only)
7. Target appliance working phase (Activity procedures only)

Procedure ID and parent procedure ID specifies the dependency structure of procedures. Type parameter specifies the procedure type: status or activity. Start time is relative to the parent's timer. For some procedures, such as using a printer, start time is not fixed; the start probability specifies the chance a procedure is started execution at any the current moment. End time, similar to start time, is relative to the parent's timer. When the start time is not fixed to a range of time, the end time usually is not fixed as well; the duration is used as trigger to finish procedures: when the timer of current procedure exceeds duration, the procedure is ended, otherwise, continue counting.

For activities, appliances desired to be used are also specified in the parameter, with the working phase. This requires the agent has knowledge of what phase the

appliance has. The format of the profile is demonstrated by the following example of a student class agent's profile:

```
PID=1,PPID=n/a,GROUP=1,TYPE=s,MULTI=1,  
START_PARAMS=32400/3600,END_PARAMS=68400/3600
```

```
PID=2,PPID=n/a,GROUP=2,TYPE=s,MULTI=1,  
START_PARAMS=43200/1800,END_PARAMS=46800/1800
```

```
PID=3,PPID=1,GROUP=3,TYPE=a,MULTI=1,  
TARGET_APP=laptop,TARGET_MOD=working,PRIORITY=0,  
START_PARAMS=300/600,DURATION_PARAMS=n/a
```

```
PID=4,PPID=1,GROUP=4,TYPE=a,MULTI=1,  
TARGET_APP=desktop,TARGET_MOD=working,PRIORITY=0,  
START_PARAMS=300/600,DURATION_PARAMS=n/a
```

```
PID=5,PPID=4,GROUP=5,TYPE=a,MULTI=1,  
TARGET_APP=display,TARGET_MOD=working,PRIORITY=0,  
PROBABILITY=1,DURATION_PARAMS=n/a
```

```
PID=6,PPID=4,GROUP=6,TYPE=a,MULTI=10,  
TARGET_APP=printer,TARGET_MOD=working,PRIORITY=0,  
PROBABILITY=0.0001,DURATION_PARAMS=60/20
```

```
PID=7,PPID=2,GROUP=7,TYPE=a,MULTI=3,  
TARGET_APP=microwave_oven,TARGET_MOD=on,PRIORITY=0,  
PROBABILITY=0.002,DURATION_PARAMS=120/60
```

```
PID=8,PPID=1,GROUP=8,TYPE=a,MULTI=1,  
TARGET_APP=light,TARGET_MOD=working,PRIORITY=0,  
PROBABILITY=1,DURATION_PARAMS=n/a
```

There are 8 procedures in a student class agent. The first 2 procedures are status: “working time” and “having lunch”, they do not involve appliance usage but to provide a dependency for other statuses or activities only. The dependency information is given in the PPID field, as it stands for “Parent PID”. MULTI field is to specify the maximum times of the procedure run. The start time and end time parameters are specified in “mu/sigma” format. Procedure 3-8 are activities, the target appliance and its working mode are specified. Probability field is used to specify the possibility of starting the procedure at current time point when there are no start time parameters. Duration parameters are used to specify the task duration when there are no end time parameters, it is specified in “mu/sigma” format.

When an activity is eligible to start, the agent sends a request for using specified appliance, the format of request pasted into the environment possesses parameters as:

1. Agent ID
2. Desired appliance
3. Desired working mode
4. Desired working time

2.2.3.5. Profiling for agent

These parameters are externally decided in a profiling procedure, the separation of data and simulator program is for the convenience in future expansion.

We surveyed individual users' behaviors (in our case, student type users in lab). And make a standard profile, and based on the standard profile, generate profiles for individual agents in simulation environment. The workflow is shown in figure 2.35:

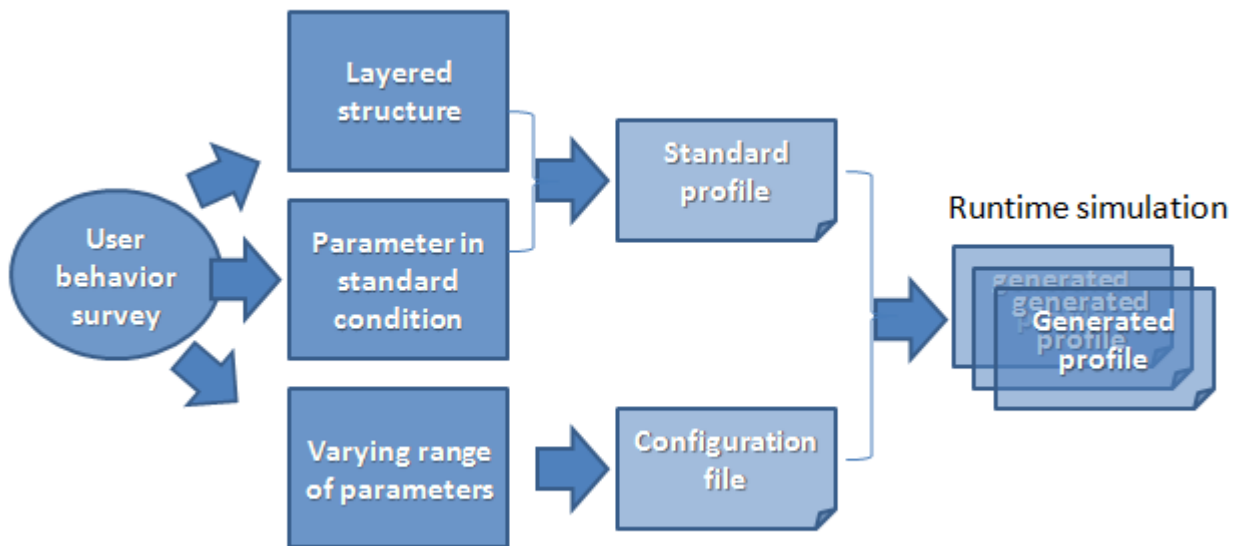


Fig.2.35.Profile making process

2.2.4. Implementation

The simulation environment consists of several modules as listed, and shown in figure 2.36:

1. Initiation module, getting input from configuration file and generate runtime profiles for later use
2. Request generating module, use agent profiles and agent simulator to generate request logs for all the agents
3. World simulator module, make instances for appliances using appliance profiles and appliance simulator, and pass the requests to them
4. Visualization module, visualize data output

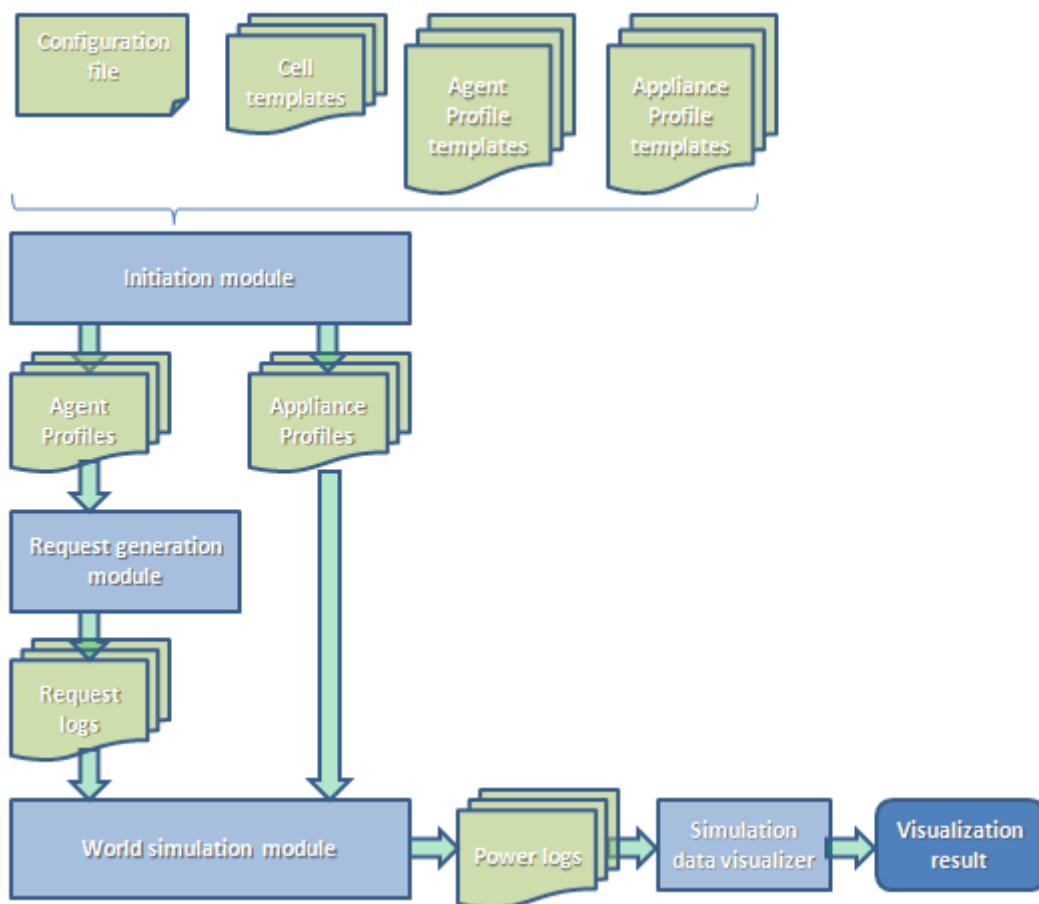


Fig.2.36 Modules and work flow

In the initiation step, the simulation environment (SE) reads parameters such as time length of the whole simulation, scale in cells types and quantities from an initiation configuration file. SE decides the types and quantities of all the necessary appliances and agents according to the configuration file of cells (cell templates). Then SE takes standard appliance and agent profiles (templates) of the decided type and their configuration files as input, and generates the runtime profiles for appliances and agents in appropriate number in the initiation step. The configuration files are used to guarantee different appearances for appliances and agents by generating their profiles with set randomness.

In request generating step, request generator uses profiles of agents and agent simulator to create instances, and gather the output requests into a request log for later use.

In world simulator module, the request log is used as a source to pass request to appliances instances created by the world simulator. The permission control and interactions between agents and appliances are implemented in this step. If the time of a request came, the world simulator sends it to the appliance, with higher priority for a personal appliance than a shared one.

The file structure of SE is shown as below:

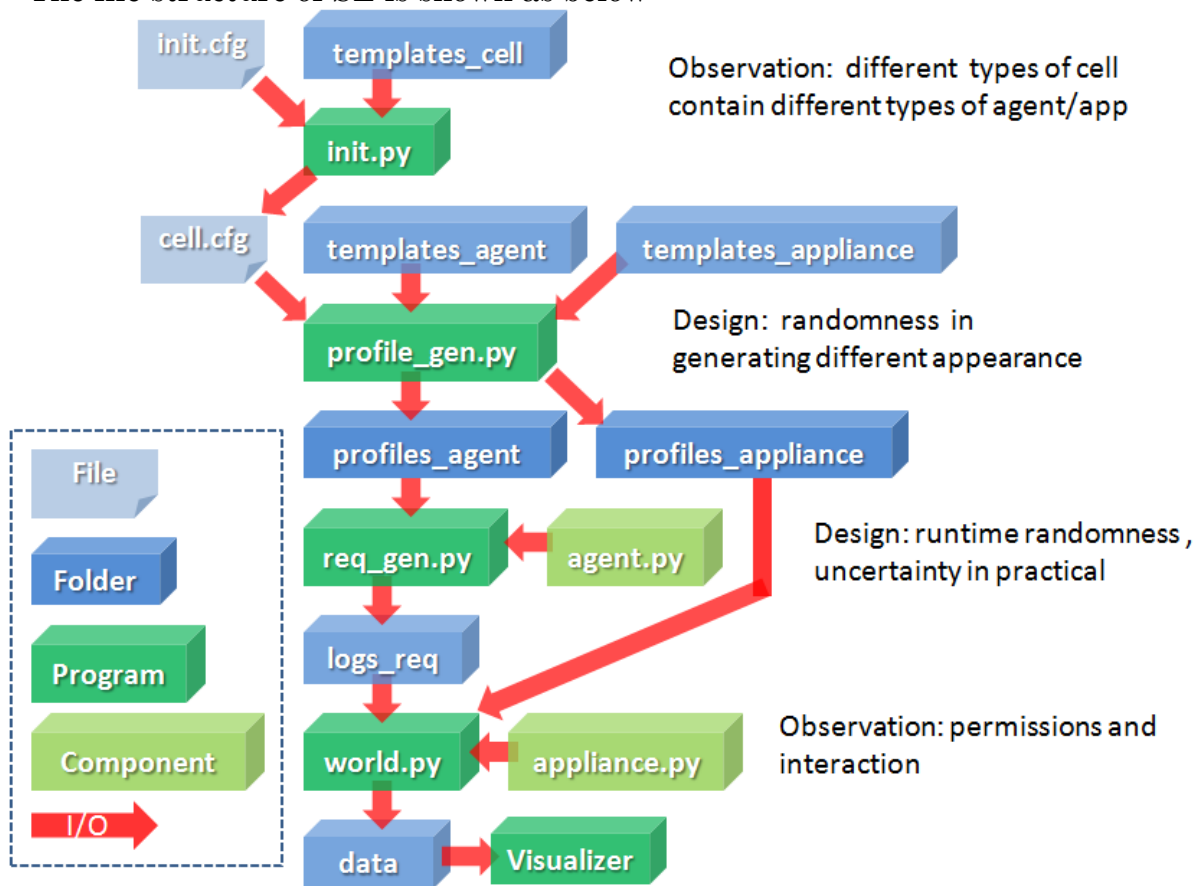


Fig.2.37.File structure

2.2.5. Result

Based on the power monitoring results, we made 13 classes of appliances' profiles: refrigerator, kettle, router, server, phone, air conditioner, microwave oven, desktop PC, laptop PC, display, printer, TV, TV recorder (Actual template profiles in Appendix). The profiles are tested in a comparison between generated simulation data and practical data.

In an example of simulation result of a research room's power during a day, is shown as below, the research room contains a number of students and common appliances, such as refrigerator, microwave ovens, desktop and laptop PCs.

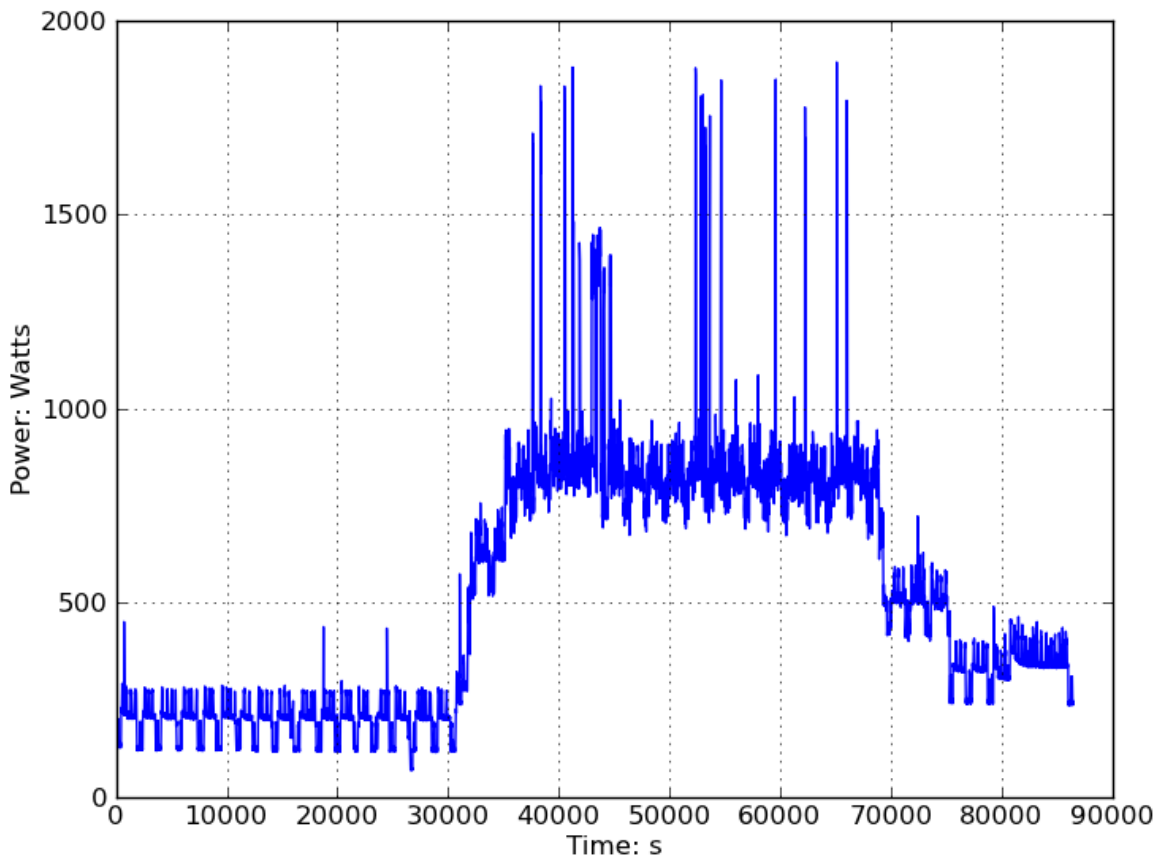


Fig.2.38. Power curve of a research room

The power curve of 100 research rooms during 10 days is shown in figure 2.39:

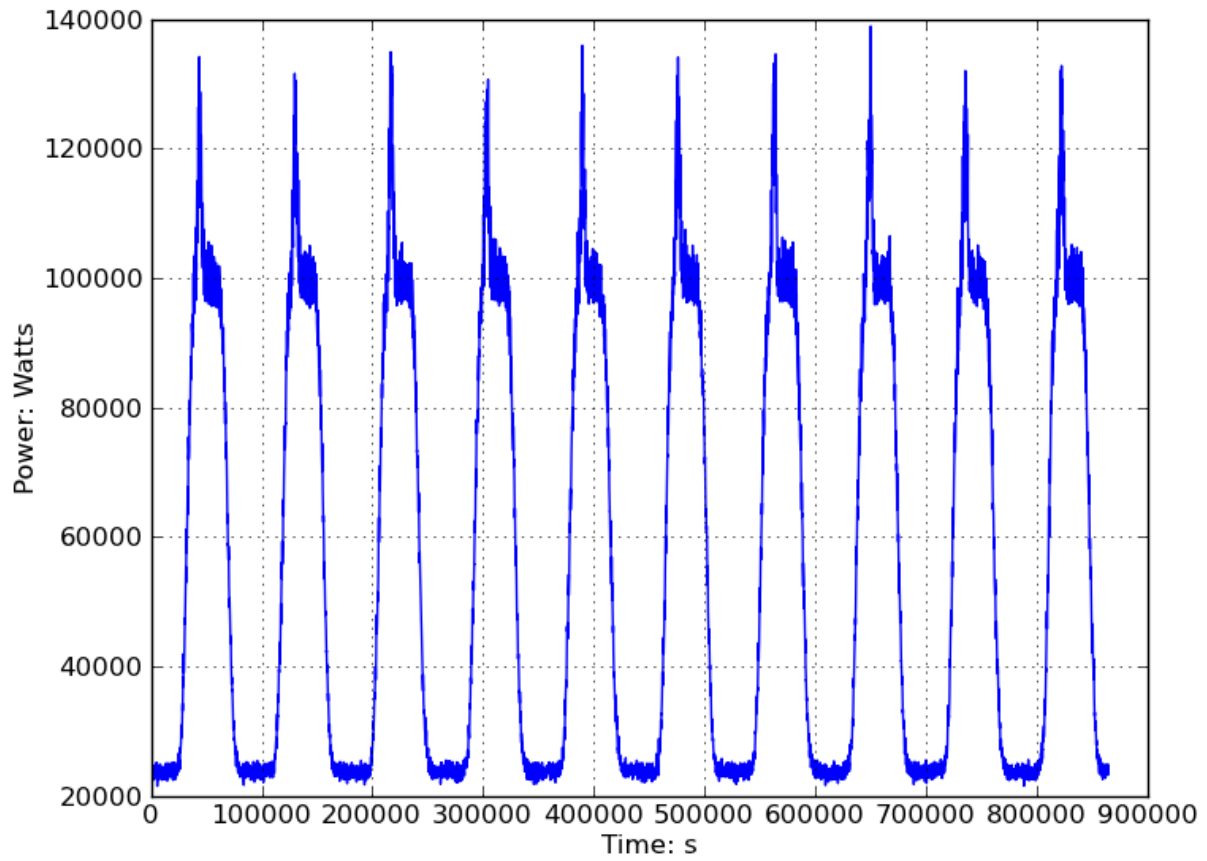


Fig.2.39. Total power of 100 research room in 10 days.

As the simulation is designed at a resolution of single appliance, the power/energy consumption of each appliance can be seen using local visualize. Figure 2.40 shows the example of each appliance's energy consumption in a room during a day.

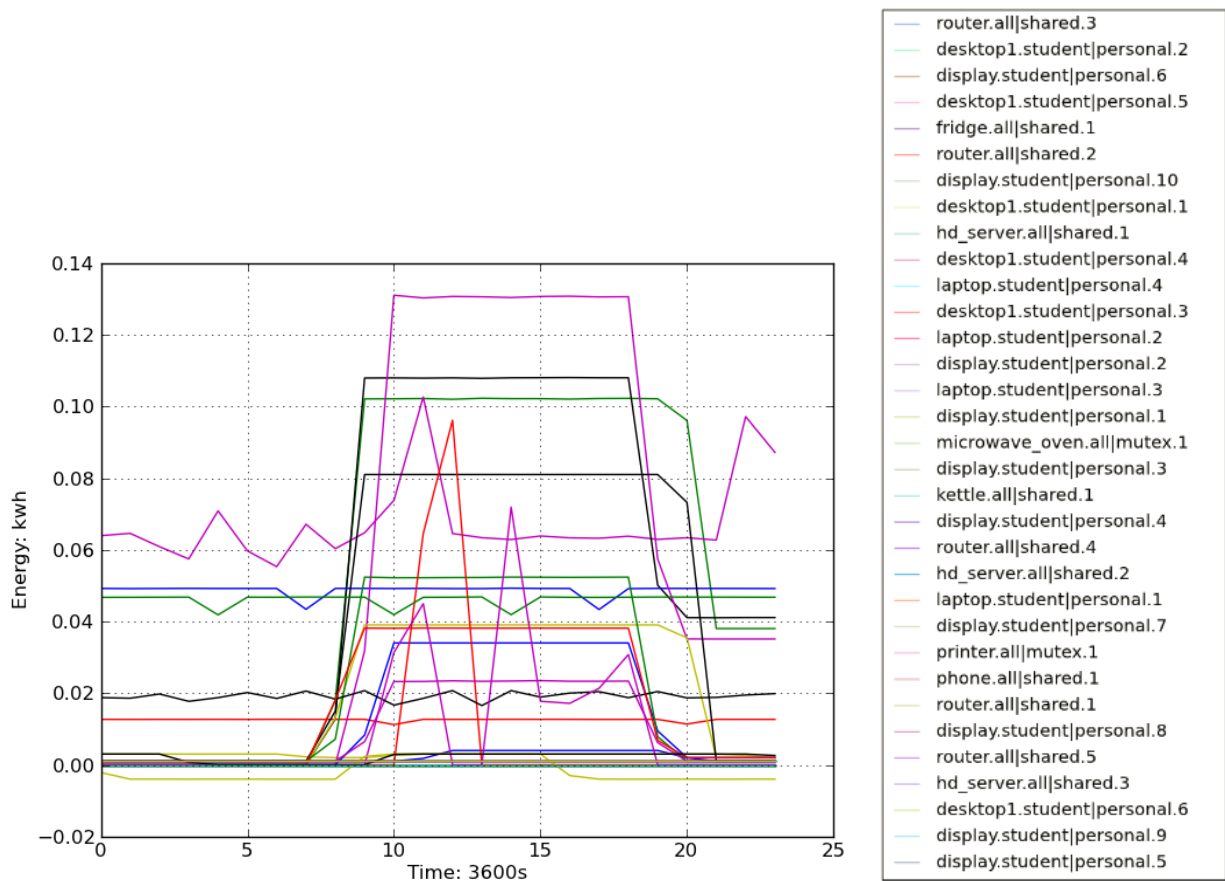


Fig.2.40. Energy consumption of each appliance in a room during a day

The accumulation of these consumptions is shown in figure 2.41.

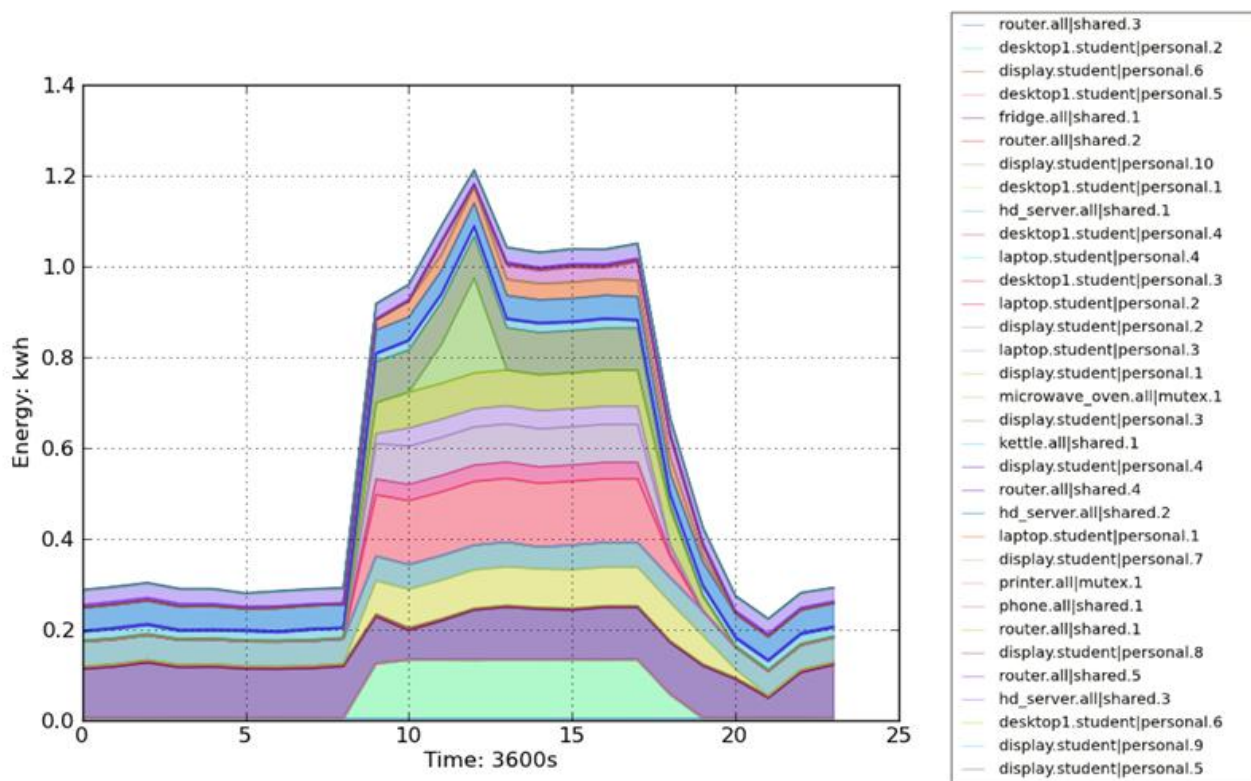


Fig.2.41. Accumulative consumption graph of appliances in a room during a day

Chapter 3. Discussion

3.1. Application of the simulation environment

In power grid, unrestrained concurrent power consumption usually causes drastic fluctuations in power curve and indirectly makes it difficult for generation to match.

Traditionally, the fluctuation is expected to be flattened mainly by massive usage from individual appliances with randomness in start time and power. The regulating mechanism is designed for eliminating the minor instability when generation works according to the guide lines made by statistic power forecasting model. This macro-scale controlling approach, does not try to eliminate time conflict usage, which is the substantial reason of the drastic fluctuation, instead, it increases tolerance of fluctuations. When the power grid is increasing in scale, and a higher stability is demanded, the approach encounters difficulties in agile responding and flexible extending.

In future smart grid, floating price can also be used as a regulating tool for balancing usage or encouraging the usage to be distributed in a desired style. This approach, in contrast to using a regulating system, aims at eliminating the fluctuation from at the usage distribution level.

Merit of using floating price as a regulating tool is that participation of consumers will increase when the usage scheduling is directly related to their own interest. The demerit, however, as the price is floating, it becomes difficult for industrial users to predict the cost.

In this part, we want to discuss the pricing policies and corresponding consumer strategies. We want to utilize the merit of floating price while try to minimize the demerit. So we proposed a simple design of floating price and consumer strategy, and we tried to use our multi-agent simulation environment to verify its effectiveness on reducing the peak height in grid power as well as the price stabilizing effect in daily usage.

3.2. Policy design on supplier end

3.2.1. Existing pricing policies

In power grid, pricing is an important feature to embody energy policy. The power price p (unit: Yen/kwh) is defined as the cost c (unit: Yen) of unit usage u (unit: kwh). Thus the cost of total usage is:

$$c = u \cdot p$$

More specifically, in a time period $[t_s, t_e]$, the usage is an integration of power over time:

$$u(t_s, t_e) = \int_{t_s}^{t_e} pwr(t) \cdot dt$$

$pwr(t)$ is the power (unit: kw) at time t .

There three basic pricing policies for deciding p , they: constant pricing, step pricing, and time-of-use (TOU) pricing.

In constant pricing the price is a constant:

$$p = k$$

That means the cost c is linear to usage u : $c = k \cdot u$

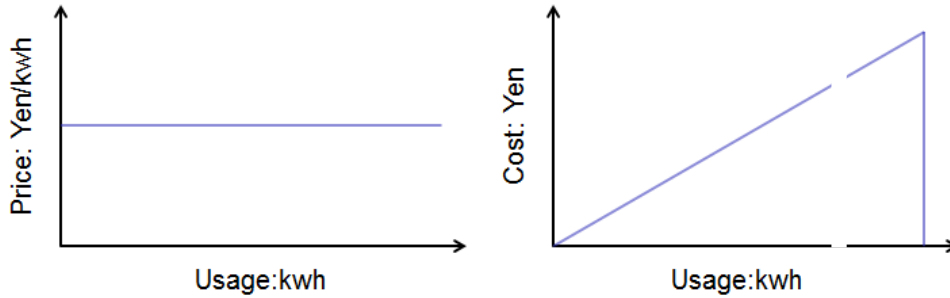


Fig.3.1. Constant pricing: price and cost

This pricing is simple and the cost is perfectly predictable, but has the short coming that: it does not put any constrain on the cost of usage from individual consumers. As the price is always the same, consumers may consume energy as much as they want.

To restrain the power consumption in the grid in a billing period of time (usually a month), step pricing is introduced. A so far usage u_s is defined as the usage from the beginning of the billing period t_0 to the current time point t_c :

$$u_s = u(t_0, t_c)$$

The price p is a piecewise function of the so far usage u_s :

$$p(u) = \begin{cases} k_0 & u_s \in [0, v_1] \\ k_1 & u_s \in (v_1, v_2] \\ k_2 & u_s \in (v_2, v_3] \\ \dots & \dots \end{cases}$$

k_i is a constant price value (unit: Yen) on usage interval $(v_i, v_{i+1}]$.

v_i is the valve value (unit: kwh) the price step changes.

Therefore, the total cost c is also a piecewise function of the so far usage u_s , as shown in figure 3.2:

$$c = \begin{cases} k_0 \cdot u_s & u_s \in [0, v_1] \\ k_0 \cdot v_1 + k_1 \cdot (u_s - v_1) & u_s \in (v_1, v_2] \\ k_0 \cdot v_1 + k_1 \cdot (v_2 - v_1) + k_2 \cdot (u_s - v_2) & u_s \in (v_2, v_3] \\ \dots & \dots \end{cases}$$

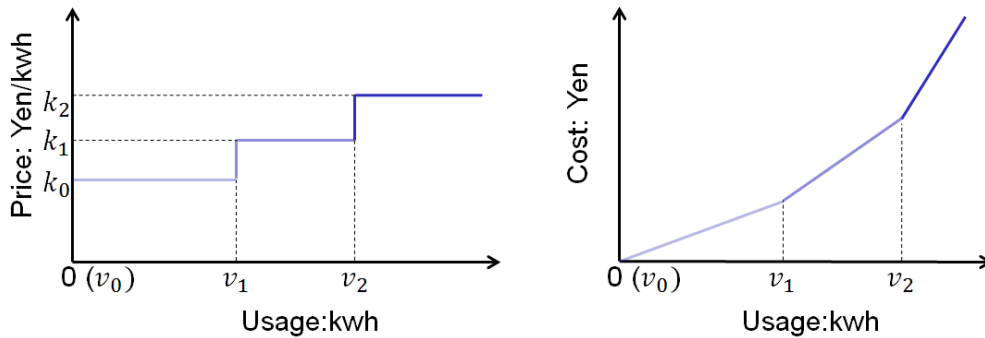


Fig.3.2. Step pricing: price (left) and cost (right), price for usage from 0 to v_1 is k_0 , price for usage from v_1 to v_2 is k_1 , price for usage from v_2 to v_3 is k_2 . The gradient of cost curve k_i is usually increasing so as to avoid power over use.

In step pricing, unit price climbs step by step when the amount of usage increases. The gradient of increasing cost is the price. As the price goes up when the usage with a billing period increase, consumer may consider not consuming all the necessary energy within one billing period, thus a relief for the grid is achieved. However, for concurrent usage, the pricing does not provide any control or guidance. Thus an intensive usage may still happen on peak hours.

To balance power usage during a day, time-of-use (TOU) pricing is introduced. The idea is to encourage more balanced power usage distribution by setting different power prices during periods of a day. Under this policy, the power cost depends not only on the amount of consumption but also the time of usage.

In TOU pricing, price p is a piecewise function of current time t_c in a day (shown in figure 3.3):

$$p(u) = \begin{cases} k_0 & t_c \in [t_0, t_1] \\ k_1 & t_c \in (t_1, t_2] \\ k_2 & t_c \in (t_2, t_3] \\ \dots & \dots \end{cases}$$

k_i is constant price value (unit: Yen) on time interval $(t_i, t_{i+1}]$.

t_i is the time point in a day from where a different pricing take place.

In TOU pricing, the cost varies not only with the amount of energy consumed, but also the time when the usage occur. The start time and end time of the pricing time period is fixed, so the consumer can predict the cost accurately according to the price table issued from power companies. Usually the price is cheaper in idling time (e.g., late night) than in the peak hours (e.g., daytime). Under such a pricing policy, consumers would consider to move some usage to periods of a cheaper price, to save some cost.

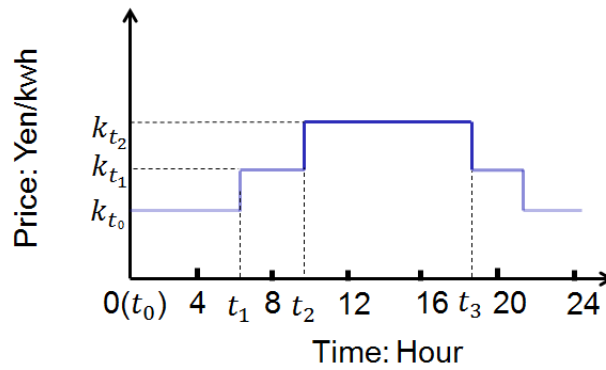


Fig.3.3. Time-of-use (TOU) pricing schematic: price varies in different time periods of a day.

In practical, modern power companies may use step pricing, TOU pricing and sometimes combined pricing policies. For example, Tokyo Electricity Power Company (TEPCO) in Japan, provides a variety of pricing plans for consumers to choose: Rate of Meter-Rate Lightening (Monthly), Time-Specific Lightening “8/10-Hour Night Service” (“Otokuna Night 8/10”), Season and time specific lightening (“Denka Jozu”).

In Rate of Meter-Rate Lightening (Monthly)pricing plan, step pricing policy is used. A residential customer’s contract will be classified as Meter-Rate Lightening B or C, the pricing table is shown as following:

Contract category			Unit	Rate(¥)
Meter-Rate Lightning B	Demand charge	10A	(per contract)	273.00
		15A	(per contract)	409.00
		20A	(per contract)	546.00
		30A	(per contract)	819.00
		40A	(per contract)	1,092.00
		50A	(per contract)	1,365.00
		60A	(per contract)	1,638.00
	Energy charge	First 120kWh (1st block rate)	1kWh	17.87
		Over 120kWh up to 300kWh (2nd block rate)	1kWh	22.86
		Over 300kWh (3rd block rate)	1kWh	24.13
	Minimum monthly charge		(per contract)	216.30
Meter-Rate Lightning C	Demand charge		1kVA	273.00
	Energy charge	First 120kWh (1st block rate)	1kWh	17.87
		Over 120kWh up to 300kWh (2nd block rate)	1kWh	22.86
		Over 300kWh (3rd block rate)	1kWh	24.13

Table 3.1. Pricing table of plan: Rate of Meter-Rate Lightning (Monthly) (Data source: TEPCO website, Aug. 2011)

Both Meter-Rate Lightning B and C are step pricing, the billing period is one month. During a month, the step prices for usage in 0-120kwh, 120-300kwh and over 300kwh are 17.87 Yen/kwh, 22.86 Yen/kwh and 24.13 Yen/kwh respectively, which means more charge for unit power consumption when the usage increase.

In Time-Specific Lightning “8/10-Hour Night Service” (“Otokuna Night 8/10”) pricing plans, the pricing policy is TOU pricing (2 pricing period) combined a step pricing policy during daytime. The difference is: in 8-Hour Service, the night time is counted from 11pm to 7am, whereas in 10-Hour Service, the night time is counted from 10pm to 8am. The detailed pricing tables are shown as following:

Time-Specific Lighting “8-Hour Night Service” (“Otokuna Night 8”)

			Unit	Rate(¥)
Demand charge	Under 6kVA		(per contract)	1,260.00
	7kVA – 10kVA		(per contract)	2,100.00
	Over 11kVA		(per contract)	2,100.00 +273.00×(Demand – 10kVA)
Energy charge	Daytime	First 90kWh (1st block rate)	1kWh	21.87
		Over 90kWh up to 230kWh(2nd block rate)	1kWh	28.07
		Over 230kWh (3rd block rate)	1kWh	29.64
	Nighttime		1kWh	9.17
Discount rate	When using appliance turned on for 5 hours		1kVA	241.50
	When using nighttime heat storage devices with controlled electricity use		1kVA	136.50
Minimum monthly charge			(per contract)	306.60

Time-Specific Lighting “10-Hour Night Service” (“Otokuna Night 10”)

			Unit	Rate(¥)
Demand charge	Under 6kVA		(per contract)	1,260.00
	7kVA – 10kVA		(per contract)	2,100.00
	Over 11kVA		(per contract)	2,100.00 +273.00×(Demand – 10kVA)
Energy charge	Daytime	First 80kWh (1st block rate)	1kWh	23.87
		Over 80kWh up to 200kWh(2nd block rate)	1kWh	30.74
		Over 200kWh (3rd block rate)	1kWh	32.48
	Nighttime		1kWh	9.48
	Discount rate	When using appliance turned on for 8 hours		1kVA
When using appliance turned on for 5 hours		1kVA	283.50	
When using nighttime heat storage devices with controlled electricity use		1kVA	178.50	
Minimum monthly charge			(per contract)	306.60

Table 3.2. Pricing table of plan: Time-Specific Lightening “8 (upper table)/10 (lower table)-Hour Night Service” (“Otokuna Night 8/10”) (Data source: TEPCO website, Aug. 2011)

The price during night time is around one-third of the price in the day. A much cheaper price encourages consumers to use electricity more in the night and reduce the consumption during the day. In day time the price policy is step pricing similar to that in the Rate of Meter-Rate Lightening (Monthly), but with a little higher charge than ordinary step pricing.

In Season and time specific lightening (“Denka Jozu”) pricing plan, the pricing policy is TOU pricing with 3 pricing periods, a detailed pricing table is shown as following:

			Unit	Rate(¥)
Demand charge	Under 6kVA		(per contract)	1,260.00
	7kVA – 10kVA		(per contract)	2,100.00
	Over 11kVA		(per contract)	2,100.00 +273.00×(Demand – 10kVA)
Energy charge	Daytime	Summer	1kWh	33.37
		Other seasons	1kWh	28.28
	Morning and evening hours		1kWh	23.13
	Nighttime hours		1kWh	9.17
Discount rate	When using appliance turned on for 5 hours		1kVA	241.50
	When using nighttime heat storage devices with controlled electricity use		1kVA	136.50
	For all-electric homes, power volume fee discounted by 5% (except summer daytime hours)			
Minimum monthly charge			(per contract)	306.60

Table 3.3. Season and time specific lightening (“Denka Jozu”) (Data source: TEPCO website, Aug. 2011)

The periods of a day in this plan are more subtly specified than in ordinary step pricing plans:

1. Daytime: from 10am-5pm
2. Morning and evening hours: 7am-10am and 5pm-11pm
3. Nighttime hours: 11pm-7am.

Since the daytime price in this plan is already high enough, it does not vary according to the power usage (step pricing), instead, it changes only when season of power consumption is different: in summer days, the price is higher than in days from other seasons.

3.2.2. Pricing function in smart grid

In future smart grid, floating pricing is used for a faster responding usage regulation mechanism in the grid. The following figure below shows a possible system configuration of future power grid.

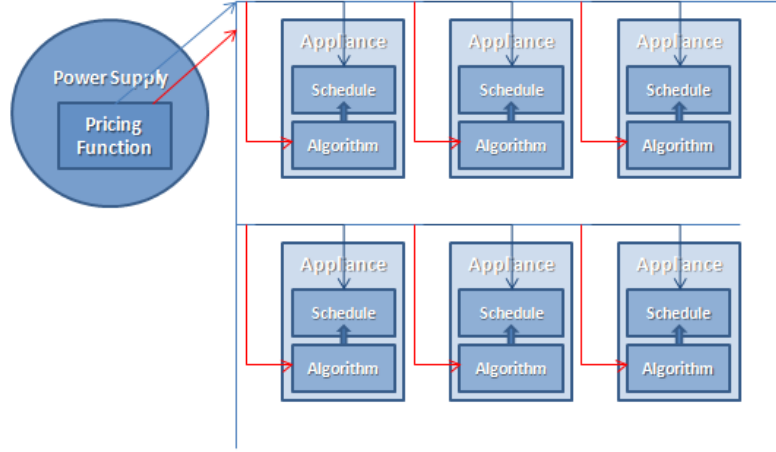


Fig.3.4. Schematic of a smart grid: Price broadcasting and task scheduling in smart grid. Blue lines are the power lines, power supplier gather power usage data by measuring the power on these lines. Red lines are the information channels, used for broadcasting price information; it may be non-physical if using existing information infrastructures.

In such a configuration, consumer end appliances act as independent agent in the grid. Doing task scheduling for non-emergent tasks according to price information received from power supplier. The appliances do not have communication with each other; the coordination of appliances is implemented using the broadcasted price information.

Similar to TOU pricing, the price in floating pricing is varying with time periods. The difference is that in floating pricing, there are more periods, for instance, there can be periods for each hour, and thus a day is divided into 24 pricing period:

$$P = (p_0, p_1 \dots p_{23})$$

P is a price vector, the components p_i are the price during hour i and $i+1$.

Different to TOU pricing, p_i is a variable, rather than a set price value. That means even a price for a specific hour may vary in value day to day. And the period of cheap power price is not fixed.

On the power supplier end, a pricing function is designed to provide such price information in the grid. Power supplier gathers past usage data of a day, and gets the price vector for the next day using the pricing function. The price vector is then sent to individual appliances for guiding the usage. This progressive procedure can be described in the following figure:

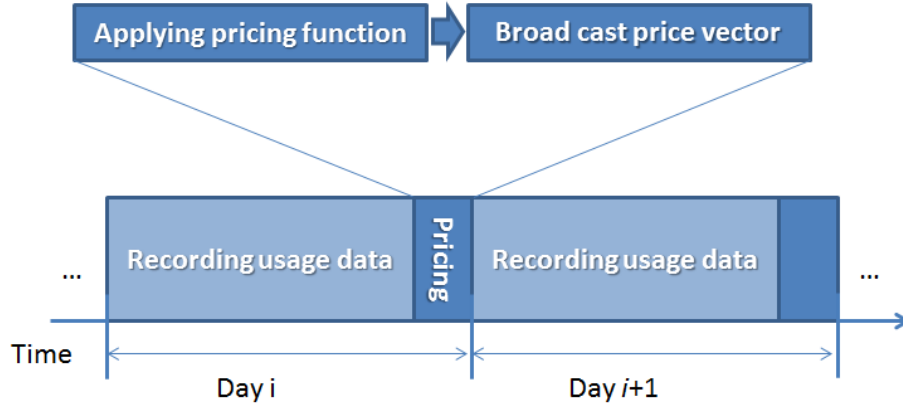


Fig.3.5. Progressive procedure of floating pricing from power supplier's view

To get the price vector with instructive information for usage scheduling, the pricing function takes the usage data U as an input:

$$U = (u_0, u_1 \dots u_{23})$$

U is a usage vector, the components u_i (unit: kwh) are the total usage in the grid during hour i and $i+1$.

Thus the pricing function F can be described as:

$$P_{i+1} = F(U_i)$$

i is the day count.

This design aims to provide not only the temporal guidance for usage, but also the usage information during periods of a day. In our experiment, we set the pricing function F a linear transform from usage vector to the price vector:

$$F : P = kU$$

k is a constant (unit: Yen/(kwh)²), in case of more complicated designs, for example, guarantee basic power supply with a low price and restrain excessive usage with a higher price, k can be set to piecewise function as in step pricing.

In this design, the price for current hour is decided by the usage from same hour in the previous day. Since the usage is dynamic, the price is floating at same time may vary in different days.

Advantage of conducting such a pricing-scheduling-pricing process is that, the system can dynamically adapt itself to changes. Practically, the fluctuating of grid power is happening all the time. The pricing should be able to re-adjust the policy to adapt to an ever-changing environment automatically, whether the time is in the day or night, winter or summer.

On consumer end, appliances receive the price vector and perform task scheduling. Floating price becomes the driving force for achieving a loose control of power usage in the grid, though it is just a supervisory guide and much freedom is left for appliances to decide their actual behaviors. Additionally, power company may artificially modify the price vector so as to achieve more complicated controls, such as making the demand somewhat follows the generation.

3.3. Task scheduling mechanism on appliance end

3.3.1. Non-emergent tasks and background usage

On consumer end, the usage balancing effect is based on the idea: if usage from a unit in certain level (like a house or a community) is balanced, the totaled usage of these units should be balanced.

More specifically, the balancing is achieved by scheduling tasks into later execution. For example, in our simulation result of power usages of a cell, we noticed that, if some usage can be scheduled from the peak hour to idling time, the total usage can be balanced as shown in the following figure:

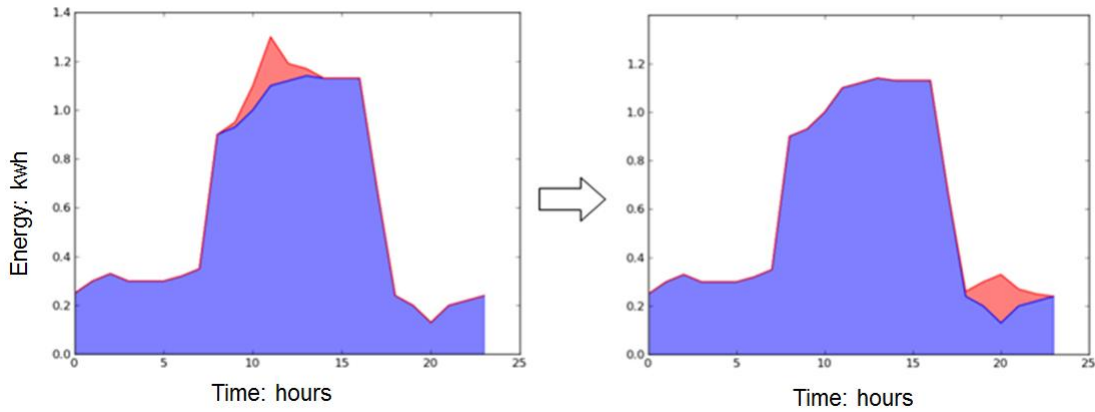


Fig.3.6. Power balancing effect of task scheduling

In practical power consuming environment, the usage of appliances that can be scheduled is usually not exactly only one in quantity; also the number of cells is normally in the scale of tens of thousands. The balancing effect would be obvious when the number of schedulable usage in each cell and the number of the cells increase.

From the view of human users, interactive appliance usages are grouped into the two types: emergent and non-emergent tasks. The task that needs immediate execution right after committed is defined as emergent task. An example is usage of a PC, when we want to use a PC, we expect the PC turns on right after the power button is pressed, so it is an emergent usage. The task that can be scheduled to later execution is defined as non-emergent task. On good candidate for such kind of task is the usage of a washing machine, when we want to use a washing machine, we usually put the laundries into the machine and leave for something else to do, usually we are not really hurry about having the work done as long as it can be done by the time we come back, so it is not emergent.

The schedulable usage mainly consists of non-emergent tasks, in a household scenario. A background usage is formed by the power consumption from emergent

tasks (non-schedulable) and the usage of non-interactive appliances. The background usage can be predicted either from monitoring of past practical power consumption data or from our simulation work.

For non-emergent tasks, the usage can be scheduled to avoid the peaks in the background usage. The design is implemented by enabling a time window for scheduling the actual start time within. The window is specified by a task commit time (usually the present in runtime, as the appliance does not know in advance when a request incomes), and a deadline (time length after commit time). The execution time of non-emergent task should be shorter than the window size so that space of scheduling is allowed. In case of emergent tasks, the time window is tightly from the commit time to the deadline, which means no scheduling is allowed, as shown in figure 3.7:

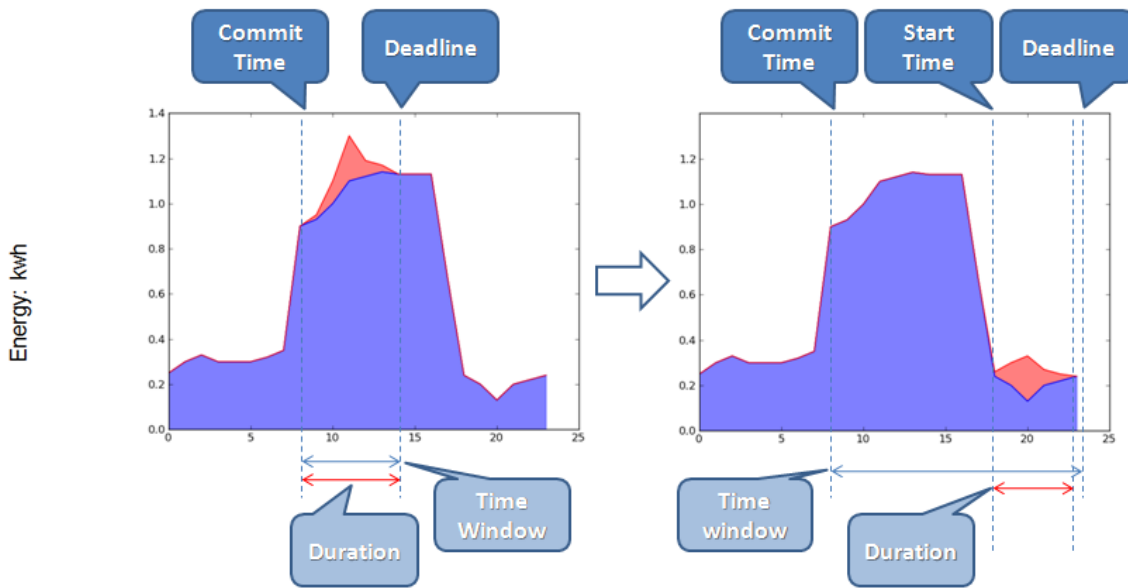


Fig.3.7. Emergent task (left) and non-emergent task scheduling within a time window (right), the contour in blue represents the background usage (sum of non-interactive appliance usage and emergent (non-schedulable) task usage), the red part is non-emergent (schedulable) task.

3.3.2. Schedule algorithm for non-emergent tasks

The usage balancing in the grid is achieved by scheduling non-emergent tasks. On individual appliances, this macro-scale effect is achieved by enabling a micro-scaling task scheduling mechanism on appliances. To make the most of the micro-scale controlling, an automatic scheduling algorithm is designed on the appliance end to perform consumer strategy. Compares to the manual task scheduling in step pricing and TOU pricing, this design means more effective cost saving for individual consumers. Meanwhile, automatic control makes the grid more responsive and resistant to power fluctuations.

This algorithm, receives the price vector from power supplier, and when a non-emergent task incomes, set the start time of actual task execution time according to the scheduling algorithm, the procedure is shown in the figure below:

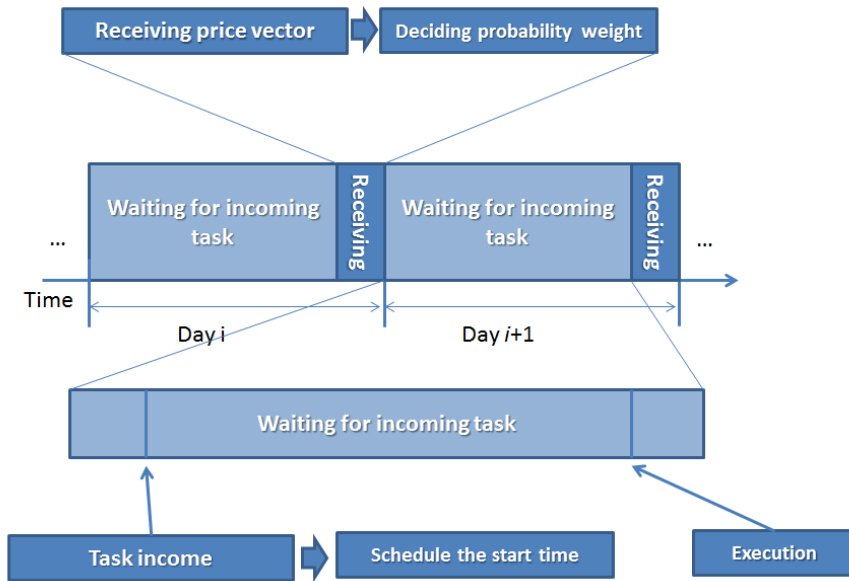


Fig.3.8. Procedure of scheduling from individual appliance's view

The power grid of appliances can be viewed as a multi-agent system (agent here refers to appliance). Each agent behaves independently to achieve their own goal in the floating pricing environment. There is no communication between agents, so no coordination explicitly exists when scheduling a task, but agents are aware of the existence of others.

From the view of appliances, the scheduling algorithm for non-emergent tasks is to help saving the cost, therefore the objectives of the algorithm are:

1. To avoid the peak hours in background usage (background usage is the usage formed from power consumption of non-interactive appliances and emergent tasks)
2. To avoid time crash usage from other agent as much as possible, when no explicit coordination exists.

To achieve the first objective, we schedule tasks following the guiding information

provided by the usage-related price vector. However, if all tasks are scheduled for the most optimized cost, their usage would crash in time and another peak usage would be created in the following day. To avoid this social disaster, which is also to achieve the second objective, we use a moderate random algorithm to decide the practical start time of a task.

In the scheduling algorithm, price vector is only guiding information. The practical start time of a task is decided with in a time window by the following procedures:

1. Get the price vector and find the greatest component p_m in the price vector:

$$p_m = \max(p_1, p_2 \dots p_{23})$$

2. Get the probability weight vector. Its components are the complementary of price vector's components to p_m :

$$PW = (p_m - p_0, p_m - p_1 \dots p_m - p_{23})$$

3. When a non-emergent task incomes, with the task start time st schedulable on a time window $[t_s, t_e]$ ($t_e = t_{dl} - t_{duration}$, t_{dl} is the deadline time of the task, $t_{duration}$ is the task duration), get normalized cumulative distribution function by integrating the probability weight function:

$$cdf(st) = \frac{\int_{t_s}^{st} PW(t) \cdot dt}{\int_{t_s}^{t_e} PW(t) \cdot dt}$$

st is the start time of task, $st \in [t_s, t_e]$.

4. Uniformly distributed random variable a is used to decide st in practical:
 $a = cdf(st)$

$$\int_{t_s}^{st} PW(t) \cdot dt = a \cdot \int_{t_s}^{t_e} PW(t) \cdot dt$$

Since the right side is a constant when the a is given, and the left side is the cumulative density function of st , which is monotonically increasing. The value of st can be uniquely decided on time window $[t_s, t_e]$. The process is described as in figure 3.9:

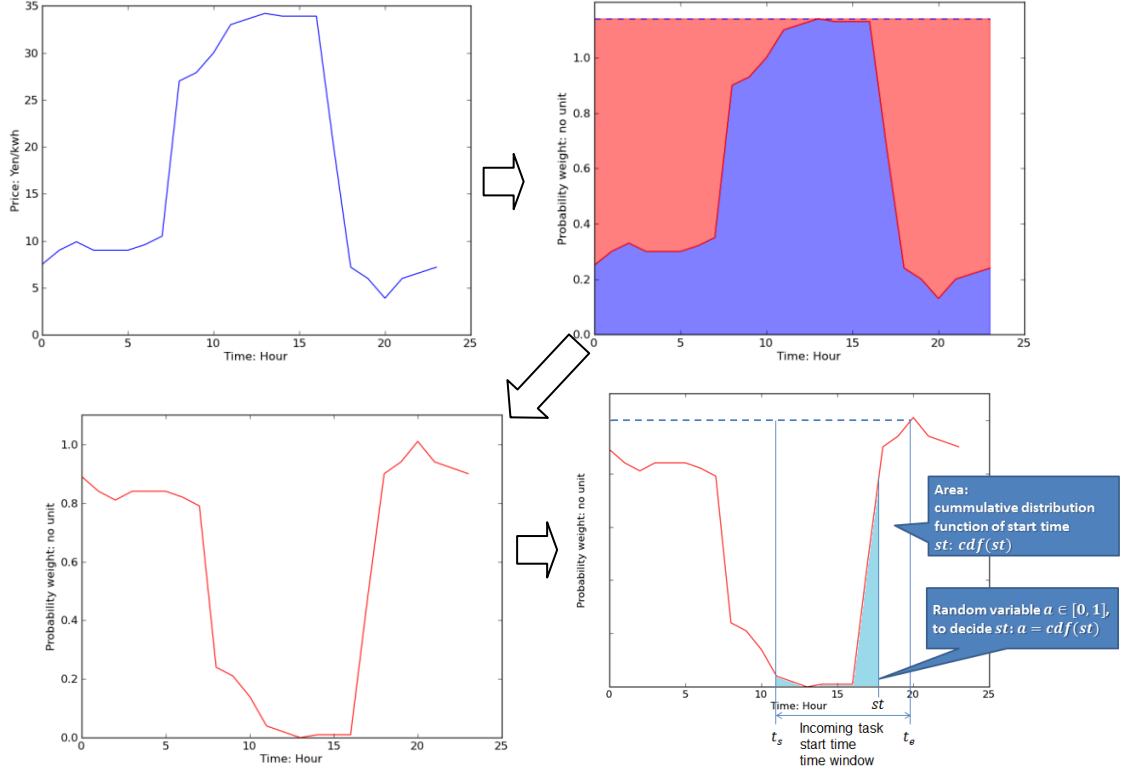


Fig.3.9. Process of deciding start time: receiving price vector (upper left), making probability weight vector (upper right and lower left), trim the time window part in the probability weight function PW as the cumulative distribution function of start time st , and use a random variable to decide the value of st .(lower right)

In this design, the complementary shape of the prices is used as a probability distribution shape. The algorithm picks the start time st according to the price vector's guiding information, this strategy guarantees the task usage to be scheduled away from the peak hours, as the probability decreases at where the price is high (also where the high usage is in linear floating pricing). The random walk on the distribution ensures the scheduling algorithm to have the best chance to avoid time crash usage with other tasks.

From the view of power supplier, by encouraging such kind of micro-scheduling mechanism, schedulable usage can be removed from peak hours and distribute the usage as evenly as possible during the idling hour, a power balancing effect can be achieved. And since the price is linear to the power consumption, if the usage is balanced, the power price would be also stabilized.

3.4. Test examples

3.4.1. Environment setting

In our approach, the pricing policy is set according to the discussion in 3.2.2, the scheduling algorithm at appliance end is designed as in 3.3.2.

In each room, there is a background power usage, this usage is from the daily constant power consumption of non-interactive appliances (such as refrigerator, electric kettle, router, etc), and the consumption of emergent tasks (such as daytime working using PC, printer, lightening, air-conditioning, etc). For simple discussion, the background usage is set as one virtual appliance, the power curve of this appliance is set to a shape of normal distribution, with basic height of 100 watts and peak height of 150 watts. The center position is around 11am (40000 second) and the width of the shape (variance) is around 1.5 hours (5000 seconds), as shown in figure 3.10.

In each room, there are also non-emergent tasks. For simple discussion, we set only one task within each room, although there can be several non-emergent tasks in real power consuming environments.

This task may represent a usage of washing machine, water heater, or charging an electric vehicle. In real power consuming environment, the power curve of background usage and schedulable tasks are usually of a more complicated shape, but for simple discussion, the task is a continuous usage of an constant power: The task duration is around 3 hours (10000 seconds), the power of the task is averagely 500 watts, the time window of task start time is around 11am (40000 second)-10pm (80000 second), as shown in figure 3.10.

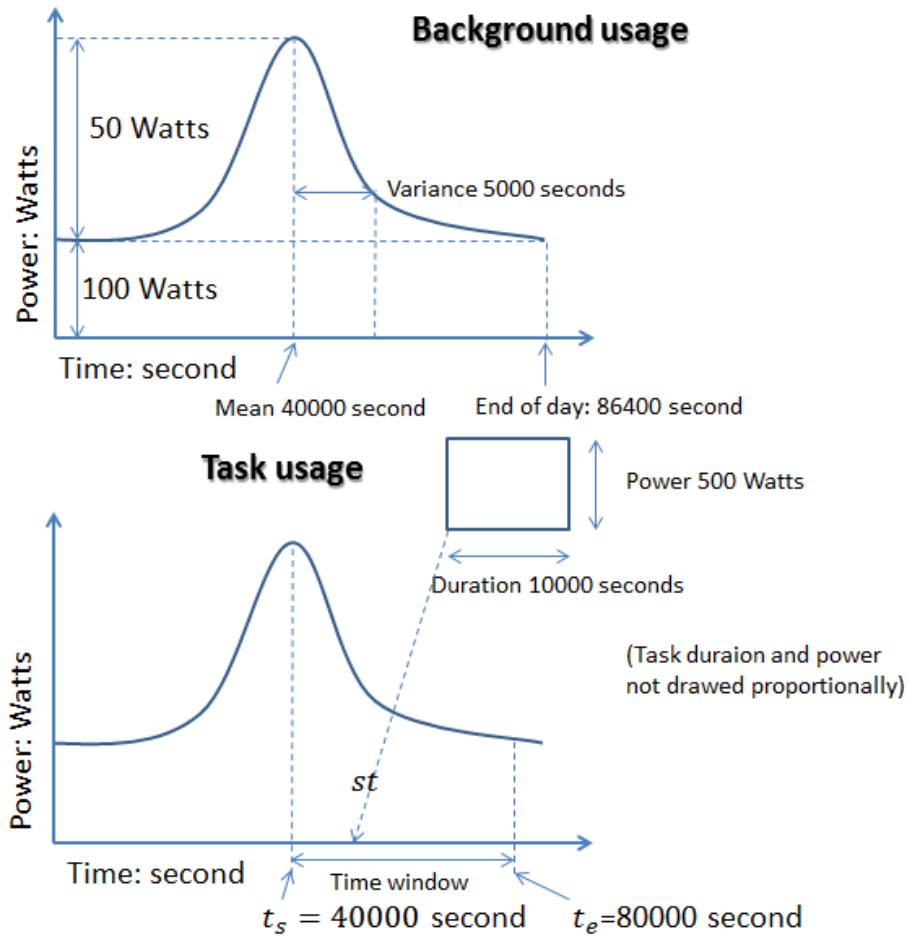


Fig.3.10. Background usage and task usage parameters

The scheduling on the appliance end is to locate the task usage onto the background usage within the task start time window, as shown. The interaction amongst appliances is achieved via power price, which is made according to total usage of all the rooms using the pricing function, and then broadcasted by the power supplier. Appliance may adjust the start time of same task in different days using the scheduling algorithm.

3.4.2. Simulation result and discussion

3.4.2.1. Test of schedulable tasks in different quantity

In the simulation of 1000 rooms in 20 days, each room has a background usage and a task, with parameter settings in 3.4.1. When all the tasks are non-schedulable, the total power of all the cells is shown in figure 3.11.

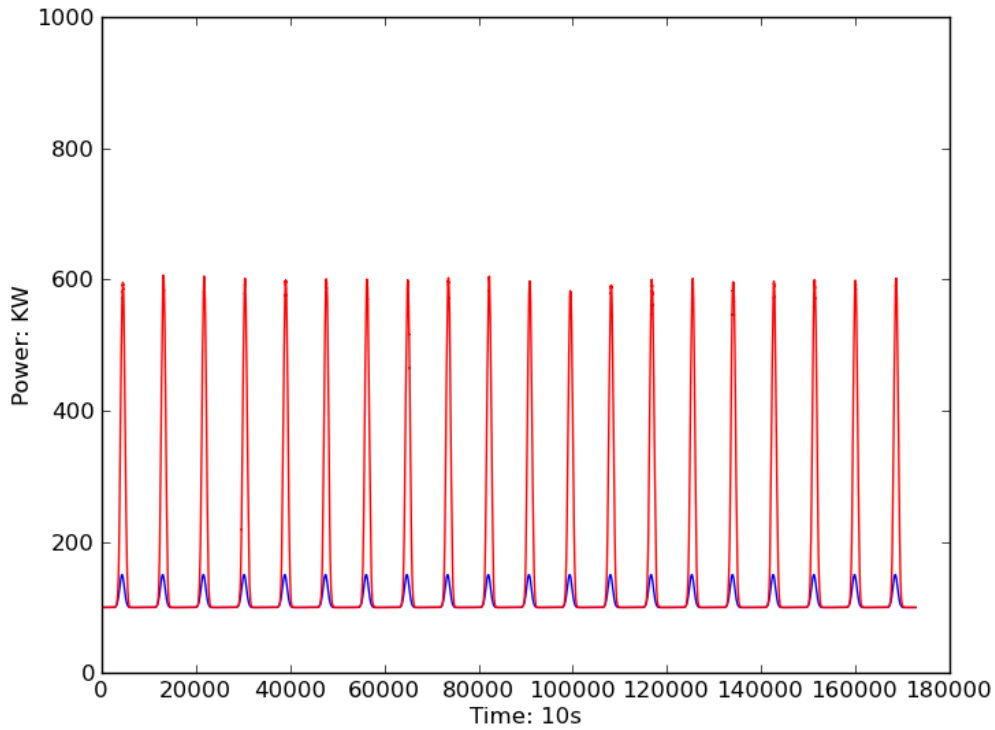


Fig.3.11. Total power including power from tasks (red line) and background power (blue line), each peak represents a daily usage

When 300 out of 1000 non-emergent tasks are scheduled by the scheduling algorithm discussed in 3.3.2, the total power would become as shown in figure 3.12.

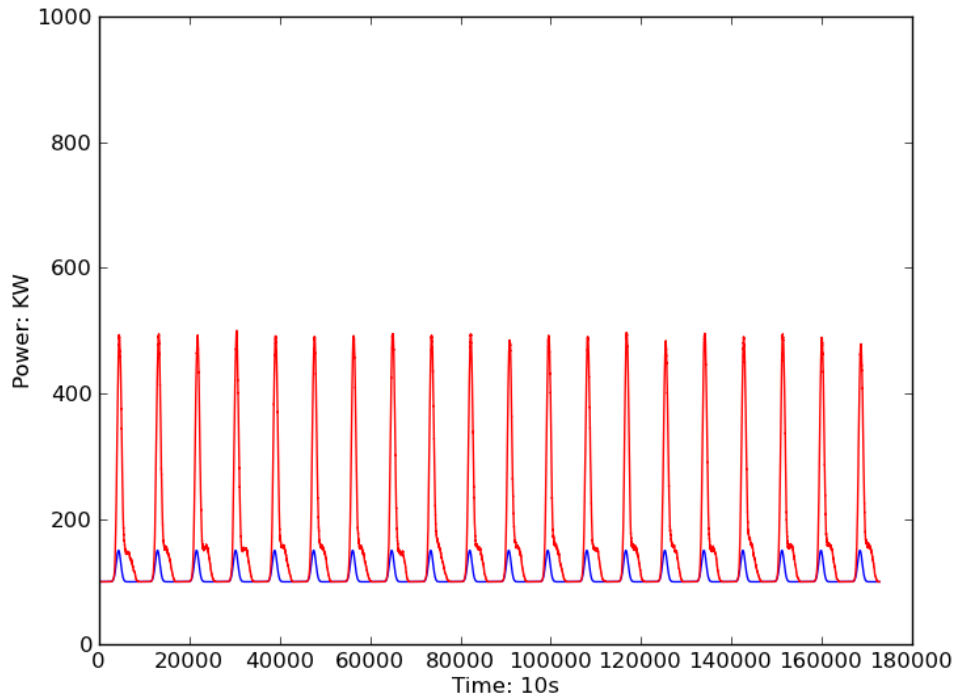


Fig.3.12. Total power (red line) and background power (blue line), each peak represent a daily usage. Schedulable tasks tend to move to into idling times.

If more tasks (600 out of 1000 tasks) became schedulable, the total power is shown in figure 3.13.

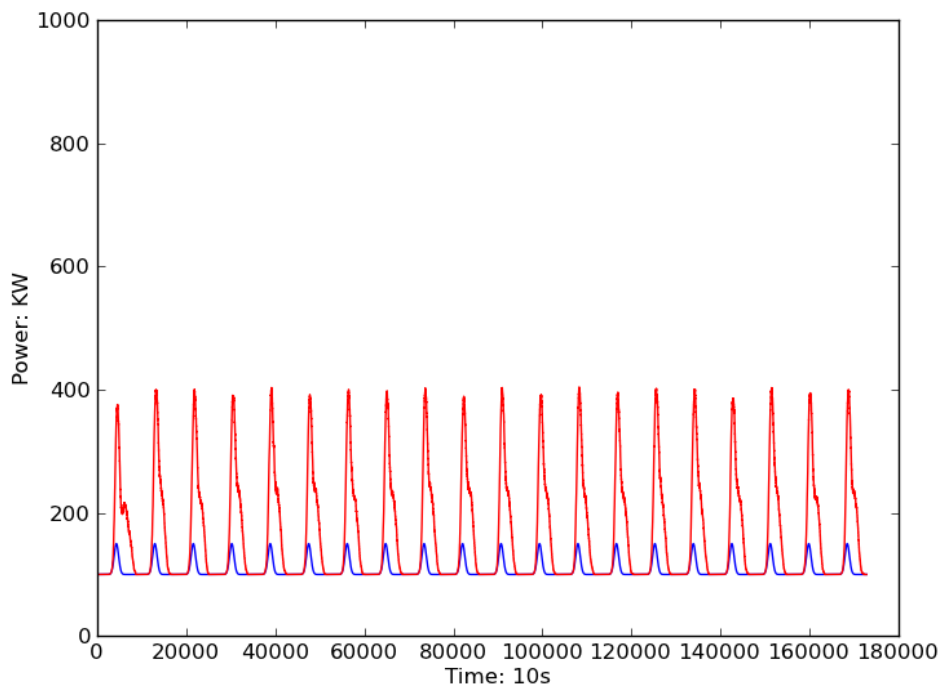


Fig.3.13 Total power (red line) and background power (blue line), each peak represent a daily usage. More tasks became schedulable, and they intend to move to idling times, which results in a reduced peak power in the grid.

If all of 1000 tasks are schedulable, the total power is shown in figure 3.14.

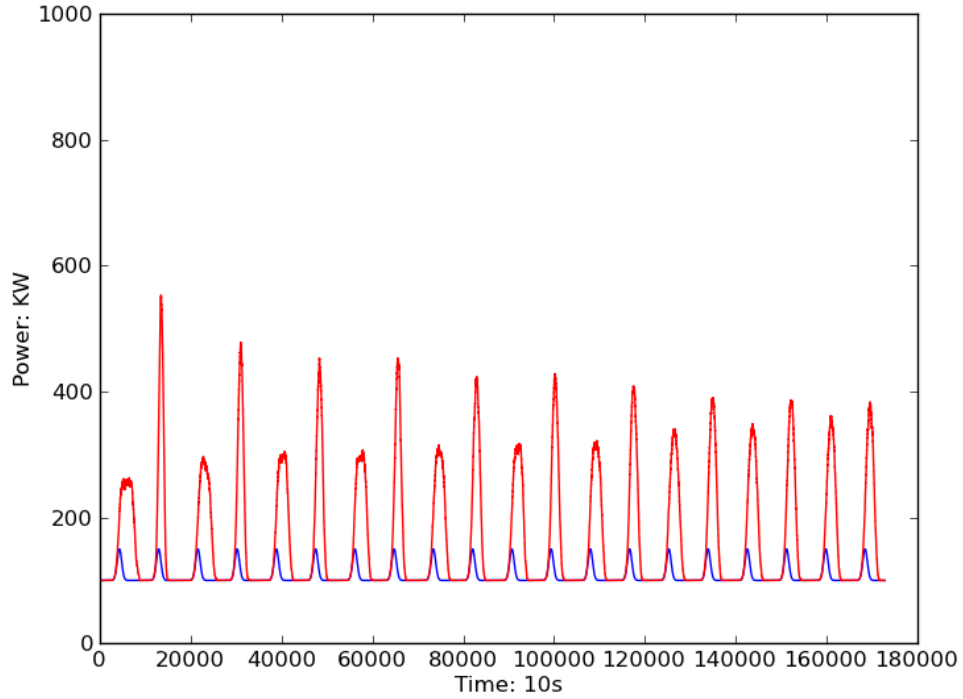


Fig.3.14. Total power (red line) and background power (blue line), each peak represent a daily usage.

From the result we see that, when the number of schedulable tasks increases, the peak power is decreasing in height. These results simulates when more and more appliances with a scheduling mechanism are put into usage, the balancing effect takes place as the non-emergent task can be scheduled into later usage.

However when all the non-emergent tasks are scheduled, the peak usage begins to fluctuate, under our direct usage-related pricing policy and appliance scheduling strategy setting.

3.4.2.2. Test of price stabilizing effect

In the previous test, we noticed the fluctuation becomes little and little in magnitude: Figure 3.15 is the power curve of 1000 rooms (with 1000 non-emergent tasks) under the same environment setting during the first 10 days:

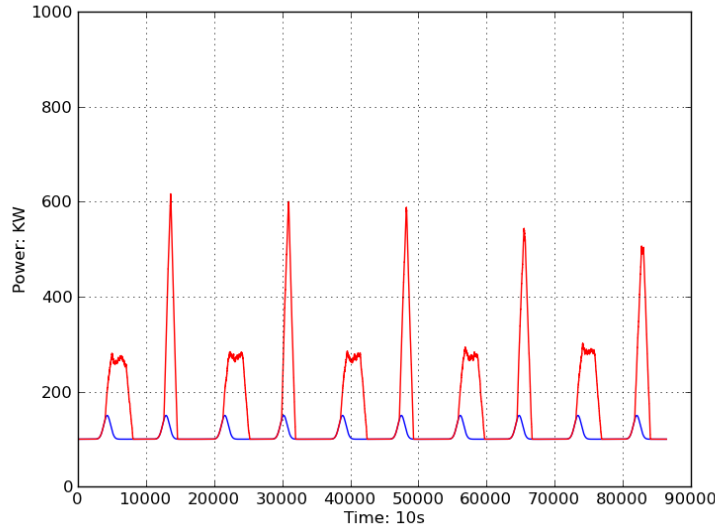


Fig.3.15. Power curve of 1000 rooms (with 1000 non-emergent tasks) during 10 days

The corresponding power usage is shown in figure 3.16.

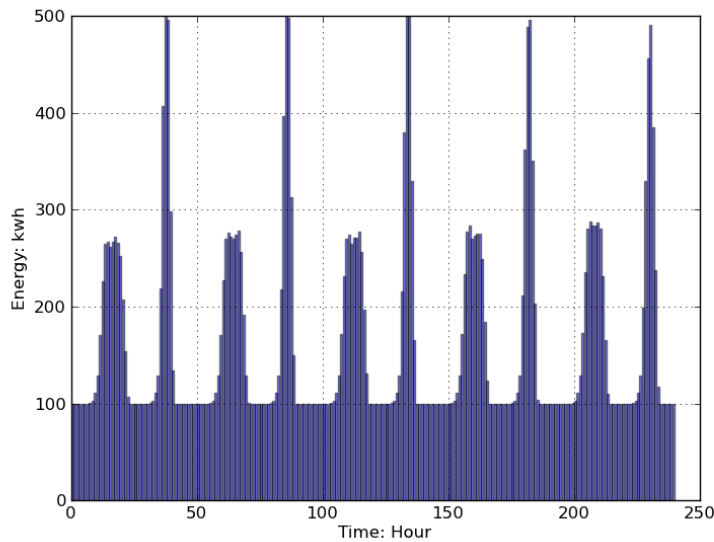


Fig.3.16. Usage fluctuation of 1000 rooms (with 1000 non-emergent tasks) during the first 10 days

In our pricing policy, the price is made according to the usage in the previous day. The price fluctuation is shown in figure 3.17. The initial price in the first day was set according the background use. Therefore the first peak looks relatively little than the following days as it does not consider usage from tasks. The pricing rate (k in our pricing function) on power supplier end in this example is set to $0.1 \text{ Yen}/(kwh)^2$, when the scale of grid increases, this rate decreases as to make the price acceptable for consumers.

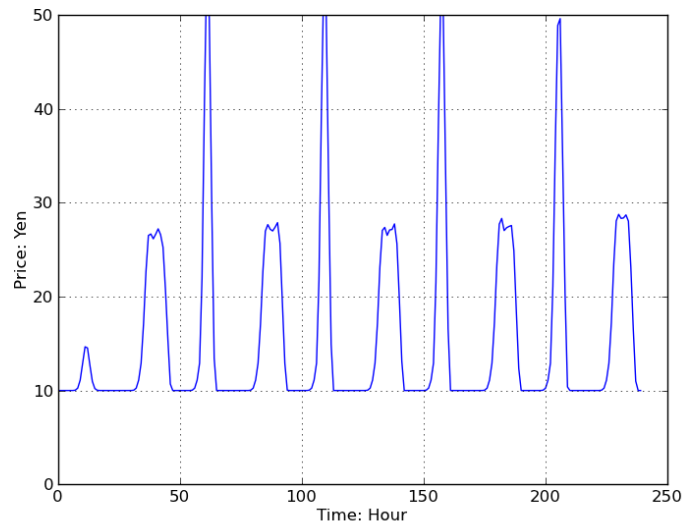


Fig.3.17. Price fluctuation during the first 10 days

The price becomes more and more stabilized when as time goes by, in the example of 30 days, as shown in figure 3.18, the power curve in the grid and in figure 3.19, the price:

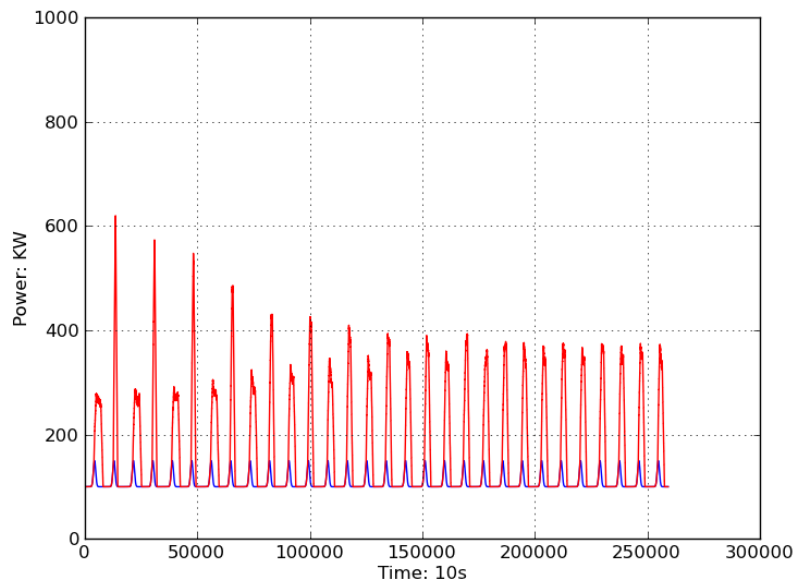


Fig.3.18. power curve of 1000 rooms (with 1000 non-emergent tasks) during 30 days

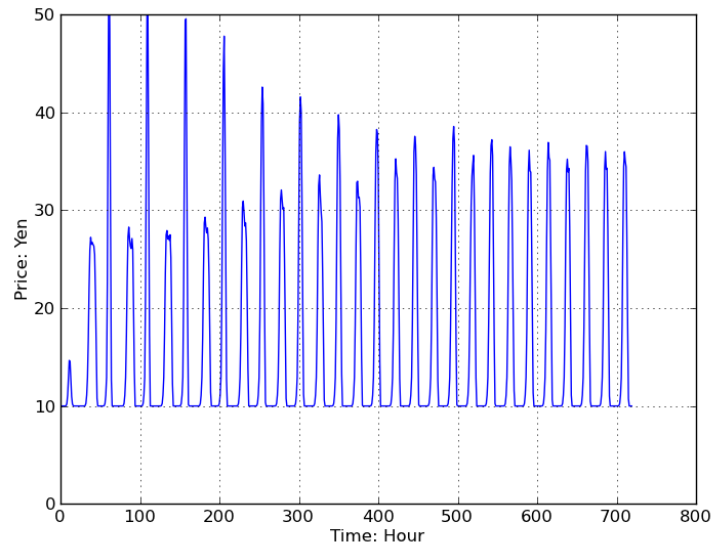


Fig.3.19. Price fluctuation in 30 days

In these figures, we see the price is being stabilized during the day of power usage. As the price goes more and more stabilized, consumers can predict the cost of their power usage, just as under normal TOU pricing.

3.4.2.3. Test of effective conditions

If the parameters of tasks and/or background changes, the proportion of usages from task to the usage from background may vary. In that case, the effectiveness of our pricing policy and scheduling algorithm may be different than in the test of 3.4.2.1 and 3.4.2.3

For instance, under the environment setting described in 3.4.1, if the task energy usage does not change in amount, when the task duration decreases, the average power of the task would increase. Figure 3.20-3.22 shows a comparison with different task parameters.

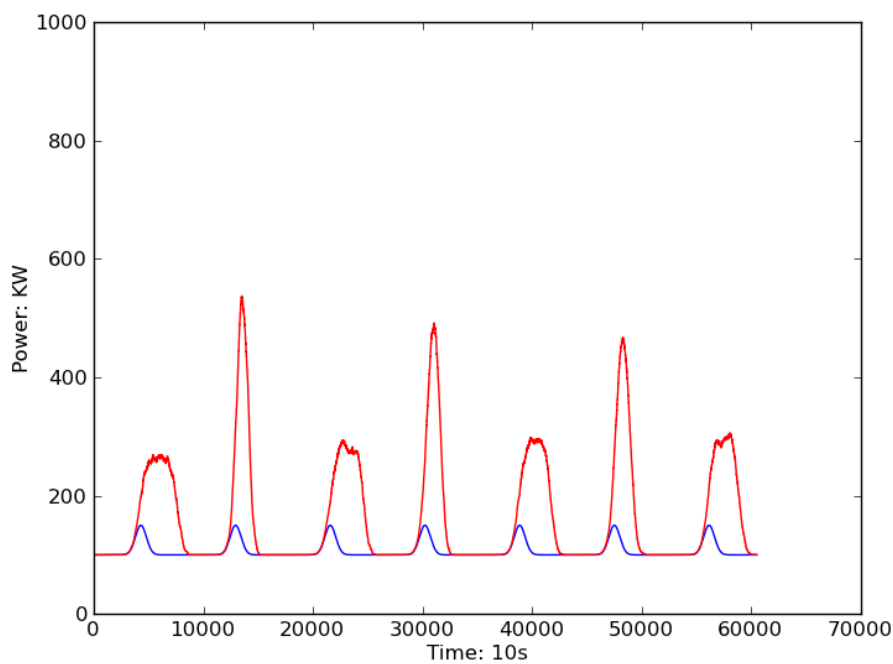


Fig.3.20. Total power curve of 1000 rooms (contains 1000 schedulable tasks) during 7 days. The task duration is 1000, average task power is 500 watts.

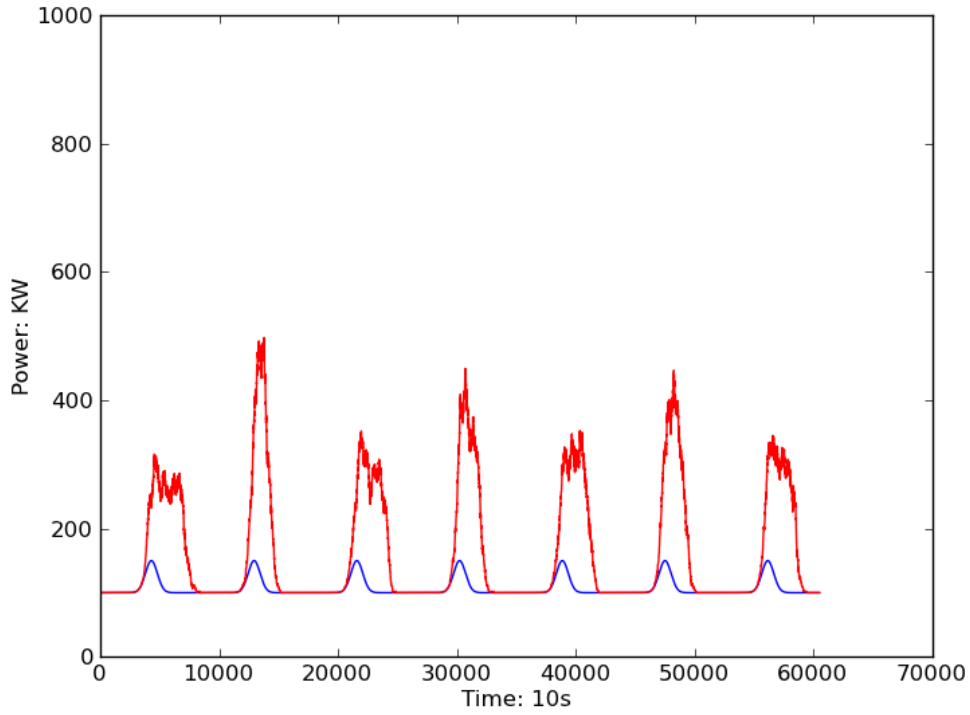


Fig.3.21. Total power curve of 1000 rooms (contains 1000 schedulable tasks) during 7 days. The task duration is decreased 250, the average task power is 2000 watts.

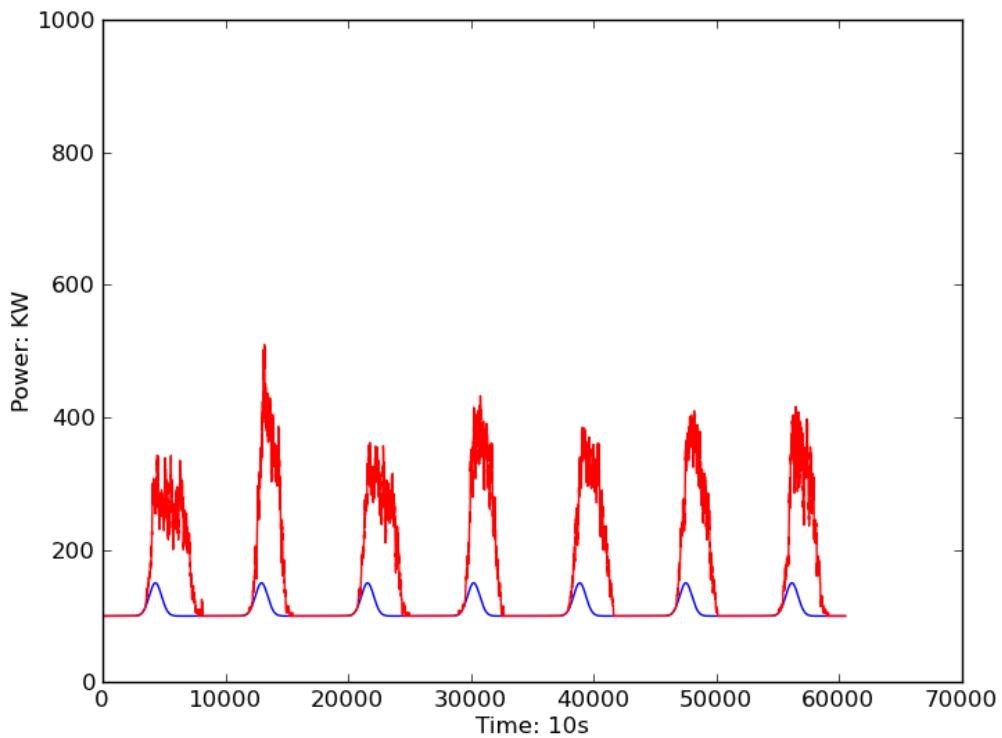


Fig.3.22. Total power curve of 1000 rooms (contains 1000 schedulable tasks) during 7 days. The task duration is decreased 100, the average task power is 5000 watts.

We noticed the fluctuation of power peak heights becomes less drastic when the task duration is decreased. As the ratio of task duration to its time window decreases, more freedom is allowed in scheduling, the tasks can have better chance to avoid time

crash with others. But in the case of decreased task duration, the power curve become not smooth when there are not enough tasks to fill the time window.

When the task duration decreases, task power has to increase if the energy demand remains the same. Currently, for the safety considering, the power of a single household task is not likely to be raised to as high as 5000 watts, but is possible in future for big power consumers like electric vehicles.

If the task duration increases, there would not be much significance in designing such task scheduling mechanism as the usage is always happen in a relatively narrow time window.

Another important thing is the ratio of usage from background to the usage from schedulable tasks. Under the same environment setting discussed in 3.4.1, to get a comparison of different task/background usage ratio, we modify the background usage, the peak value of the background usage is set from 150 (100+50) watts to 300 (100+200) watts and then to 600 (100+500) watts as shown in figure 3.23-3.25:

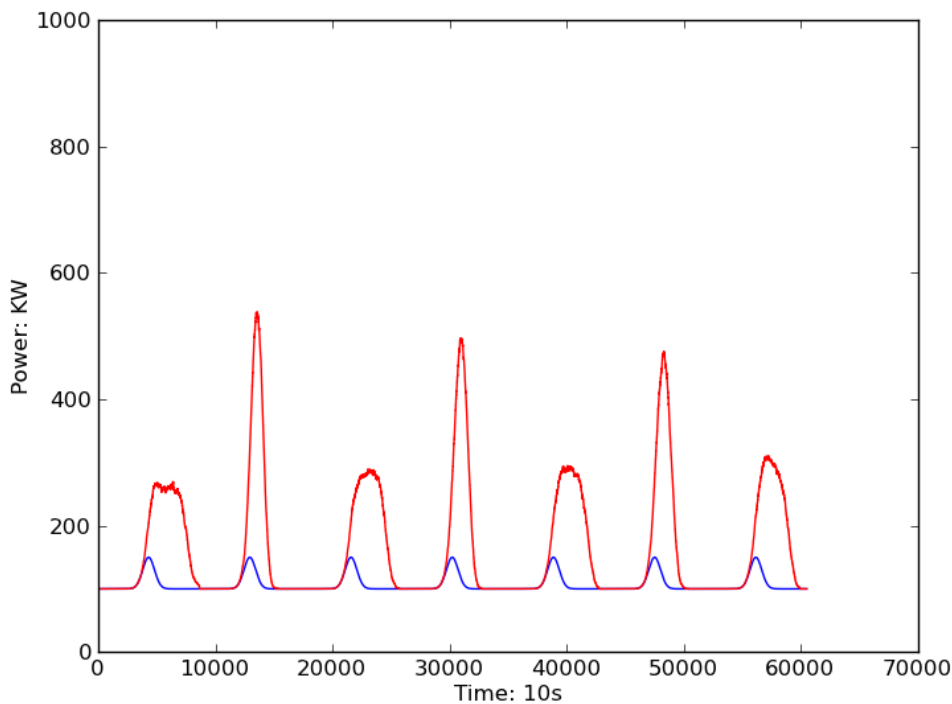


Fig.3.23. Total power curve of 1000 rooms (contains 1000 schedulable tasks) during 7 days. Background usage is of a peak value of 150 (100+50) watts

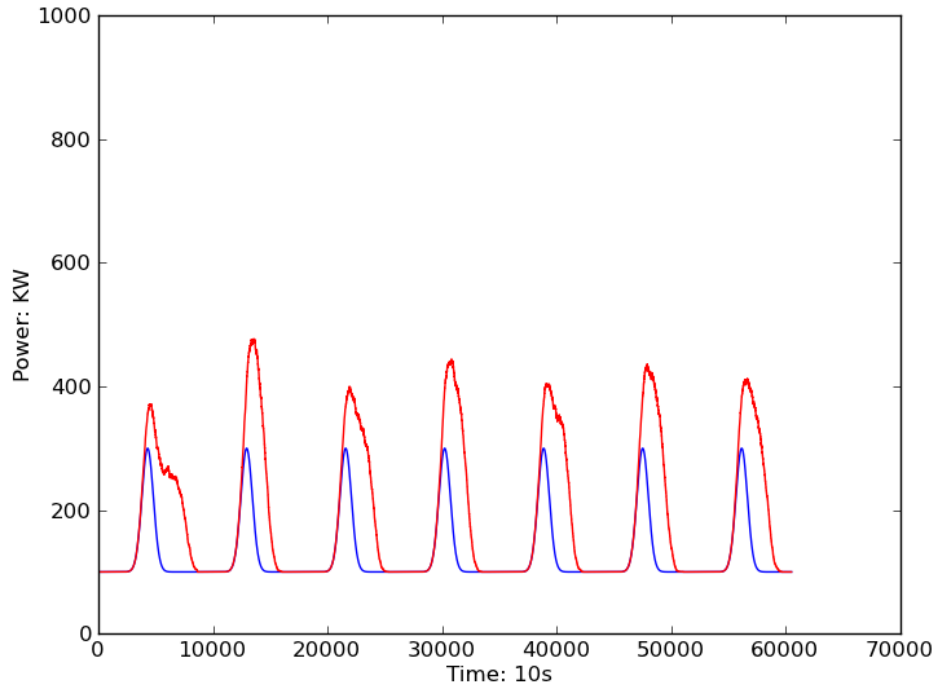


Fig.3.24. Total power curve of 1000 rooms (contains 1000 schedulable tasks) during 7 days. Background usage is of a peak value of 300 (100+200) watts

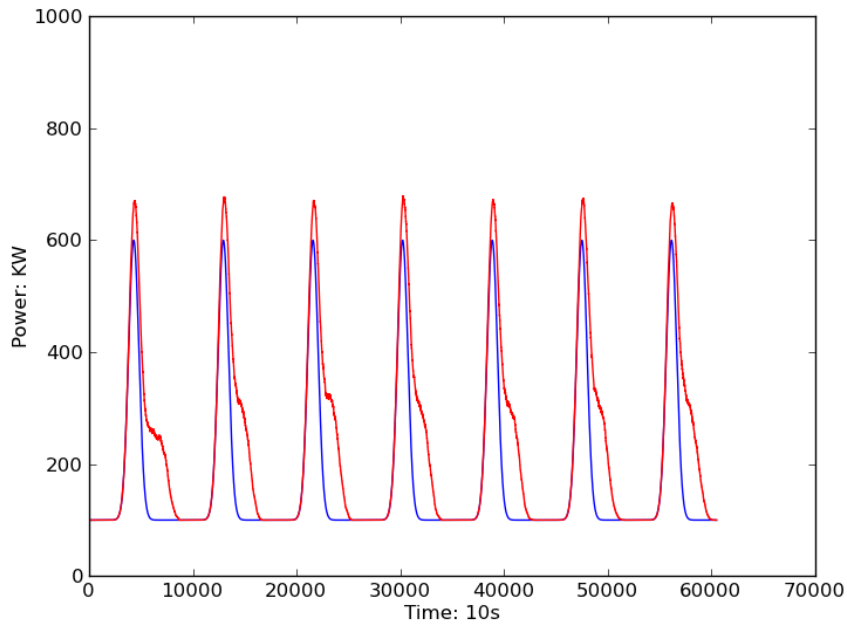


Fig.3.25. Total power curve of 1000 rooms (contains 1000 schedulable tasks) during 7 days. Background usage is of a peak value of 600 (100+500) watts

From the results we know that, when the background usage increases compares to the task usage (or the task usage decreases compares to the background usage), the scheduling is would be of little significance, as the major factor caused unbalanced usage is not the tasks, but the background usage.

On the contrary, if the task usage is much more than that from background, the effect of background usage on the usage fluctuation is relatively small. The only thing

task scheduling needs to consider about is avoid time crash usage with other tasks. A simple random walk scheduling would be sufficient enough to achieve this objective, thus there are no need to set a progressive pricing policy; a constant pricing should be the best to guide the scheduling.

From this test, we know that our pricing policy and scheduling strategy design is effective only when the usage from tasks is comparable to the usage from background. Thus we suggest that under such a pricing policy and scheduling strategy design, if there are tasks enough in number, reduce the task duration (or even divide a task into short duration sub-tasks) would be beneficial in the balancing the grid power usage. Meanwhile, this design is best effective when the task usage and background usage are comparable, when the situation is different, we need to modify the pricing policy and/or scheduling algorithm.

3.5. Summary

In this chapter, we discussed a usage-related floating pricing policy, and working with price information received, we designed a simple task scheduling mechanism on appliance end. As an application of our multi-agent system simulation environment, we discussed about this design of policy and corresponding strategy, by examining the usage balancing effect in the virtual grid:

1. In the first test, we see effectiveness of the scheduling mechanism under a floating pricing policy: when more non-emergent tasks become schedulable using the scheduling algorithm built in the smart appliances, the grid power can be balanced.
2. In the second test, we see under floating pricing policy, the fluctuation of grid power peak heights, decreases in magnitude along with time passage. In our power-related pricing, the stabilizing effect is also shown in power prices, which is crucial for consumers to budget their energy usage cost.
3. In the third test, we examined the effective conditions of our policy/strategy setting. The method is best effective when the usage from schedulable tasks and usage from background are comparable. When the number of schedulable tasks is great enough, long time window for scheduling task execution is recommended.

Chapter 4. Conclusion

4.1. Research summary

As a beginning of our work, we developed a power monitoring system using ready-made smart metering devices and database software components. Using this system, we are able to accurately record the power measures of specific types of appliances and monitor their usage in real time. The system is well-functioning and sufficient for performing power data collecting tasks.

In the power consumption simulation part, we proposed a multi-agent system modeling approach. We view the power grid as a set of cells containing two types of entities: appliance simulators and user agents. Interaction between human users such as permission and blockage in competitive usage are considered, as well as the interaction between human users and appliances. On implementation, we model appliance simulator as state machines that exhibits different working phases on its external power behavior. For human users, we model human behavior as a layered structure of procedures, agent simulator generate requests according its own internal procedure table.

We apply a profiling mechanism for enabling the appliance and agent simulator to behave as different class of appliance and human user. We make the profiles according the practical data captured by our power monitoring system, and the agent profile according to user behavior surveys. This design enables a more flexible parameter setting for the multi-agent system, and adapt to different environment setups by modifying the overall composition and adjusting the behaviors for individual appliances and human user agents in their profiles.

Randomness is introduced to generate more appearances of appliances and agents in their profiles, as well as bringing some uncertainty on runtime.

The simulation environment design is competent in modeling a power consumption environment in any scale, and is able to provide the power consuming data at a resolution of individual appliance level.

On the application layer, the simulation environment can be used as testing platform for some advanced features of smart grid. As an example, we discussed the advantage of bringing micro-scale task scheduling mechanism in balancing the grid power usage under the floating pricing policy. The system is proven a useful tool for analyzing these features.

4.2. Conclusion

We developed a power monitoring system based on the advanced metering devices. Using existing software component, we designed a real time power data capturing and storing mechanism. It is shown that this method to be applicable in building a database system for a power grid.

Multi-agent system is a useful approach for building a flexible and expansible power consumption simulation system in a large scale. With appropriate profiling, the system is able to perform power consuming simulation at a resolution of individual appliances.

Using the simulation environment built this way, we confirmed the effectiveness of floating pricing policy and corresponding scheduling mechanism in balancing grid power usage. We may design, and test quantitatively the advanced features of future smart grid, such as grid pricing policy and appliance end strategy. Multi-agent system approach is suitable not only in building a power forecasting system but also a policy/strategy testing platform as well.

4.3. Future work

Our work aims to discuss about a multi-agent system approach for building a more realistic and flexible power consumption simulation environment. The simulation environment built this way is expected to be used not only a consumption forecasting tool, but also a platform for discuss about pricing policy designing and consumer strategy making in the grid.

Before putting into practical use, a careful profiling of the appliances' power behavior and human agents' activities is needed. Detailed profiles may guarantee a more accurate simulation result.

Multi-agent system controlling methods can be applied in a context of smart grid using our simulation environment. A further research about pricing policies under different situations may be conducted with verification from simulation data. On the appliance end, a more effective scheduling algorithm considering detailed task information can be researched, as a micro-scale controlling problem.

Also, this work provides a possible way to anticipate the emerging of new appliances and distribute energy resources: their impacts on power consuming environment and grid policy/strategy making.

Chapter 5. References

1. Demand dispatch, A. Brooks, E. Lu, D. Reicher, C. Spirakis, B. Wehl, Power and Energy magazine, IEEE, Vol.8.Issue 3, 20-29p
2. Scalable Multi-Agent System for Real-Time Electric Power Management, L. M. Tolbert, H. Qi, F. Z. Peng, Power Engineering Society Summer Meeting, 2001, IEEE, Vol.3, 1676-1679p
3. Grid of the future, A. Ipakchi, F. Albuyeh, Power and Energy magazine, IEEE, Vol.7.Issue 2,52-62p
4. Power Load Management as a Computational Market, F. Ygge, H. Akkermans, the 2nd International Conference on Multi-Agent System ICMAS'96
5. Operation of a Multiagent System for Microgrid Control, A. L. Dimeas, N. D. Hatziargyriou, Power Systems, IEEE Transactions on, Vol.20.Issue 3, 1447-1455p
6. Agent based Micro Grid Management System, J. Oyarzabal, J. Jimeno, J. Ruela, A. Engler, C. Hardt, Future Power Systems, 2005 International Conference on
7. Distributed Control Concepts using Multi-Agent technology and Automatic Markets: An indispensable feature of smart power grids, M. P. F. Hommelberg, C. J. Warmer, I. G. Kamphuis, J. K. Kok, G. J. Schaeffer, Power Engineering Society General Meeting, 2007, IEEE, 1-7p
8. Multi-Agent System in a Distributed Smart Grid: Design and Implementation, M. Pipattanasomporn, H. Feroze, S. Rahman, Power Systems Conference and Exposition, 2009. PSCE '09. IEEE/PES, 1-8p
9. Whole-House Measurements of Standby Power Consumption, P. Bertoldi, A. Ricci, A. T. de Almeida, Energy efficiency in household appliances and lighting, 278-285p
10. Household electricity end-use consumption: results from econometric and engineering models, B. M. Larsen, R. Nesbakken, Energy Economics Vol.26, Issue 2, 179-200p
11. Principles of Compositional Multi-Agent System Development, F. M. T. Brazier, C. M. Jonker, J. Treur, Proc. of the IFIP'98 Conference IT&KNOWS'98.
12. Residential end-use energy simulation at city scale, Y. Shimoda, T. Fujii, T. Morikawa, M. Mizuno, Building and Environment Vol.39, Issue 8, 959-967p
13. Functions of a Local Controller to Coordinate Distributed Resources in a Smart Grid, A. Chuang, M. McGranaghan, Power and Energy Society General Meeting - Conversion and Delivery of Electrical Energy in the 21st Century, 2008 IEEE,1-6p
14. Dynamic Pricing and Stabilization of Supply and Demand in Modern Electric Power Grids, M. Roozbehani, M. Dahleh, S. Mitter, Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on,543-548p
15. Optimal Real-time Pricing Algorithm Based on Utility Maximization for Smart Grid, P. Samadi, A. Mohsenian-Rad, R. Schober, V. W. S. Wong, J. Jatskevich, 415-420p
16. Optimal and Autonomous Incentive-based Energy Consumption Scheduling Algorithm for Smart Grid, A.-H.Mohsenian-Rad, V. W. S. Wong, J. Jatskevich, R. Schober, 1-6p

Chapter 6. Appendix

In this appendix, the contents are:

Part A: Source code of PMS

Part B: Source code of SE

Part C: Sample monitoring result of 13 classes of appliances

Part D: Template profiles for the 13 classes of appliances

In part A, the files attached are:

- ops_db.py
- ops_table.py
- init_db.py
- init_tables_data.py
- init_table_registry.py
- util.py
- update_tables_data.py
- update_table_registry.py
- inquiry.py
- power_single.py
- practical_visualizer.py

The privacy information about the DBMS, such as username and password is replaced by “*****”.

In part B, the files attached are:

- appliance.py
- agent.py
- init.py
- profile_gen.py
- req_gen.py
- world.py

In part C, the monitored appliances are:

- refrigerator
- kettle
- microwave oven
- router
- server
- phone
- air conditioner
- desktop PC
- laptop PC
- display
- printer
- TV
- TV recorder

In part D, template profiles for each appliance listed in part C are attached in the same order as in part C.

Part A: Source code of PMS

ops_db.py

```
import MySQLdb
#dbms ops
def connect(hostname,username,pswd):
    return MySQLdb.connect(hostname,username,pswd)
def lst(connection):
    cursor=connection.cursor()
    cursor.execute("""SHOW DATABASES""")
    lst=[]
    for i in cursor.fetchall():
        lst.append(i[0])
    return lst
def select(connection,dbn):
    sql="""USE %s""" % dbn
    connection.cursor().execute(sql)
#db ops
def create(connection,dbn):
    sql="""CREATE DATABASE %s""" % dbn
    connection.cursor().execute(sql)
def drop(connection,dbn):
    sql= """DROP DATABASE %s""" % dbn
    connection.cursor().execute(sql)
```

ops_table.py

```
import ops_db
#db ops
def lst(connection,dbn):#connection: db must be selected
    ops_db.select(connection,dbn)
    cursor=connection.cursor()
    cursor.execute("""SHOW TABLES""")
    lst=[]
    for i in cursor.fetchall():
        lst.append(i[0])
    return lst
#table ops
def create(connection,tbn):
    sql="""CREATE TABLE %s (datetime DATETIME NOT NULL, val INT)""" % tbn
    connection.cursor().execute(sql)
def reg_create(connection,tbn):
    sql="""CREATE TABLE %s (owner CHAR(50),app CHAR(50),ch CHAR(50),location
CHAR(50))""" % tbn
    connection.cursor().execute(sql)
def drop(connection,tbn):
    sql="""DROP TABLE %s""" % tbn
```

```

        connection.cursor().execute(sql)
#data ops
def write(connection,tbn,datetime,val):
    sql="""DELETE FROM %s WHERE (datetime='%s')""" % (tbn,datetime)
    connection.cursor().execute(sql)
    sql = """INSERT INTO %s (datetime, val) VALUES ('%s', '%d')""" %
(tbn,datetime,val)
    connection.cursor().execute(sql)
def read(connection,tbn,st,et):
    sql="""SELECT * FROM %s WHERE datetime>='%s' AND datetime<='%s'""" %
(tbn,st,et)
    cursor=connection.cursor()
    cursor.execute(sql)
    return cursor.fetchall()

```

init_db.py

```

#params
hostname="localhost"
username="*****"
pswd="*****"
dbn="powerdb"

#script
import ops_db
connection=ops_db.connect(hostname,username,pswd)
if dbn not in ops_db.lst(connection):
    ops_db.create(connection,dbn)
    print "Database %s added" % dbn
else:
    print "%s already exist" % dbn
connection.close()

```

init_tables_data.py

```

#params
hostname="localhost"
username="*****"
pswd="*****"
dbn="powerdb"
mac_address="mac_address_data.csv"

#util
def csv2lst(fn):#csv->lst
    l=[]
    for i in open(fn,'r').readlines():
        l.append(i.split(',')[1])
    return l
def lst2chlst(l):#lst->chlst
    chlst=[]

```

```

    for i in l:
        chlst.append(i+ "_ch1")
        chlst.append(i+ "_ch2")
        chlst.append(i+ "_ch3")
        chlst.append(i+ "_ch4")
    return chlst

```

```

#script
import ops_db
import ops_table
connection=ops_db.connect(hostname,username,pswd)
if dbn in ops_db.lst(connection):
    ops_db.select(connection,dbn)
    for i in lst2chlst(csv2lst(mac_address)):
        if i not in ops_table.lst(connection,dbn):
            ops_table.create(connection,i)
            print "Table %s added" % i
        else:
            print "Table %s already exist" % i
else:
    print "Database %s not exist" % dbn
connection.close()

```

init_table_registry.py

```

#params
hostname="localhost"
username="*****"
pswd="*****"
dbn="powerdb"
registry="owner_app_ch_loc"

#script
import ops_db
import ops_table
connection=ops_db.connect(hostname,username,pswd)
if dbn in ops_db.lst(connection):
    ops_db.select(connection,dbn)
    if registry not in ops_table.lst(connection,dbn):
        ops_table.reg_create(connection,registry)
        print "Table %s added" % registry
    else:
        print "Table %s already exist" % registry
else:
    print "Database %s not exist" % dbn
connection.close()

```

util.py

```

#make file list

```

```

from datetime import datetime, timedelta
def datespan(start,end,delta):
    sl,el=start.split(),end.split()
    sy,smon,sd=int(sl[0].split("-")[0]),int(sl[0].split("-")[1]),int(sl[0].split("-")[2])
    sh,smin,ss=int(sl[1].split(":")[0]),int(sl[1].split(":")[1]),int(sl[1].split(":")[2])
    ey,emon,ed=int(el[0].split("-")[0]),int(el[0].split("-")[1]),int(el[0].split("-")[2])
    eh,emin,es=int(el[1].split(":")[0]),int(el[1].split(":")[1]),int(el[1].split(":")[2])
    l=[]
    current=datetime(sy,smon,sd,sh,smin,ss)
    while current<datetime(ey,emon,ed,eh,emin,es):
        l.append(current)
        current+=timedelta(seconds=delta)
    return l
def file_lst(start,end,url_dir):
    l=[]
    for i in datespan(start,end,3600):
        year=str(i.year)
        if i.month<10:
            month="0"+str(i.month)
        else:
            month=str(i.month)
        if i.day<10:
            day="0"+str(i.day)
        else:
            day=str(i.day)
        if i.hour<10:
            hour="0"+str(i.hour)
        else:
            hour=str(i.hour)
        l.append("%s/%s/%s/%s/%s-%s.xml" %
(url_dir,year,year+month,year+month+day,year+month+day,hour))
    return l
#retrieve file (file2lst)
import urllib
def retrieve(url):
    l=urllib.urlopen(url).readlines()
    if l[0]=="<powerlog>\n":
        l.pop()
        l.pop(0)
        return l
    else:
        print "%s not found" % url
        return 0
#parse (record2dic)
def xml_parser(str):
    dict={}
    start,end=str.index("<src>"),str.index("</src>")
    dict["src"]=str[start+5:end]
    start,end=str.index("<uid>"),str.index("</uid>")
    dict["uid"]=str[start+5:end]

```



```

start,end=str.index("<date>"),str.index("</date>")
dict["date"]=str[start+6:end]
start,end=str.index("<time>"),str.index("</time>")
dict["time"]=str[start+6:end]
start,end=str.index("<type>"),str.index("</type>")
dict["type"]=str[start+6:end]
start,end=str.index("<tempC>"),str.index("</tempC>")
dict["tempC"]=str[start+7:end]
start,end=str.index("<humidity>"),str.index("</humidity>")
dict["humidity"]=str[start+10:end]
#power measure
start,end=str.index("<ch1><watts>"),str.index("</watts></ch1>")
dict["ch1"]=str[start+12:end]
start,end=str.index("<ch2><watts>"),str.index("</watts></ch2>")
dict["ch2"]=str[start+12:end]
start,end=str.index("<ch3><watts>"),str.index("</watts></ch3>")
dict["ch3"]=str[start+12:end]
start,end=str.index("<ch4><watts>"),str.index("</watts></ch4>")
dict["ch4"]=str[start+12:end]
if dict["ch1"]=="":
    dict["ch1"]="0"
if dict["ch2"]=="":
    dict["ch2"]="0"
if dict["ch3"]=="":
    dict["ch3"]="0"
if dict["ch4"]=="":
    dict["ch4"]="0"
dict["ch1"]=int(dict["ch1"])
dict["ch2"]=int(dict["ch2"])
dict["ch3"]=int(dict["ch3"])
dict["ch4"]=int(dict["ch4"])
return dict

```

update_tables_data.py

```

#!/usr/bin/python

#params
hostname="localhost"
username="*****"
pswd="*****"
dbn="powerdb"
url_dir="*****"
delta=93600#86400+7200
delta2=7200

#script
#relative time: for updating
import time
from datetime import datetime, timedelta

```

```

t=time.localtime()
cur_t=datetime(t[0],t[1],t[2],t[3],t[4],t[5])-timedelta(seconds=delta2)#today.3am,      if
executed at 5am
start,end=(cur_t-timedelta(seconds=delta)).isoformat('          '),cur_t.isoformat('
')#yesterday.1am-today.3am
#absolute time: for initializing
#start,end="2011-02-19 08:00:00","2011-02-20 12:00:00"
import ops_db
import ops_table
connection=ops_db.connect(hostname,username,pswd)
ops_db.select(connection,dbn)
import util
for fn in util.file_lst(start,end,url_dir):
    if util.retrieve(fn)==0:
        continue
    for record in util.retrieve(fn):
        dict=util.xml_parser(record)
        uid=dict["uid"].upper()
        t=dict["date"]+' '+dict["time"]
        ops_table.write(connection,uid+'_ch1',t,dict["ch1"])
        print "%s updated at %s" % (uid+'_ch1',t)
        ops_table.write(connection,uid+'_ch2',t,dict["ch2"])
        print "%s updated at %s" % (uid+'_ch2',t)
        ops_table.write(connection,uid+'_ch3',t,dict["ch3"])
        print "%s updated at %s" % (uid+'_ch3',t)
        ops_table.write(connection,uid+'_ch4',t,dict["ch4"])
        print "%s updated at %s" % (uid+'_ch4',t)

```

update_table_registry.py

```

#params
hostname="localhost"
username="*****"
pswd="*****"
dbn="powerdb"
registry_fn,sheet_name='registry.xls','registry'
registry_tbn='owner_app_ch_loc'

#script
import xlrd
sheet=xlrd.open_workbook(registry_fn).sheet_by_name(sheet_name)
import ops_db
import ops_table
connection=ops_db.connect(hostname,username,pswd)
if dbn in ops_db.lst(connection):
    ops_db.select(connection,dbn)
    sql="""DELETE FROM %s""" % registry_tbn
    connection.cursor().execute(sql)#delete duplicated entries
    for i in range(1,sheet.nrows):#insert updated entries for each device
        raw=sheet.row_values(i)

```

```

        mac_address,location=raw[0],raw[1]
    for j in range(2,6):
        if raw[j]=="":
            continue
        else:
            app_info=raw[j].split(".")
            owner=app_info[0]
            app_type=app_info[1]
            if len(app_info)==2:
                app_id=""
            elif len(app_info)>=3:
                app_id=app_info[2]
            sql="""INSERT INTO %s (owner,app,ch,location) VALUES
('%s','%s','%s','%s')""" %
(registry_tbn,owner,app_type+'.'+app_id,mac_address+"_ch"+str(j-1),location)
            connection.cursor().execute(sql)
            print "Table owner_app_ch_loc updated"
    else:
        print "Database %s not exist" % dbn
    connection.close()

```

inquiry.py

```

import ops_db
import ops_table
def ch(connection,ch,st,et):
    return sort(ops_table.read(connection,ch,st,et))
def sort(t):
    dict={}
    for i in t:
        dict[i[0]]=i[1]
    l=dict.keys()
    l.sort()
    ll=[]
    for i in l:
        ll.append((i,dict[i]))
    return ll
#util
import time
import datetime
def str2datetime(s):
    date,time=s.split(' ')[0],s.split(' ')[1]
    year,month,day=date.split('-')[0],date.split('-')[1],date.split('-')[2]
    hour,minute,second=time.split(':')[0],time.split(':')[1],time.split(':')[2]
    return
datetime.datetime(int(year),int(month),int(day),int(hour),int(minute),int(second))
def scale_generate(s,e,interval):#s,e:datetime str.
    st,et=str2datetime(s),str2datetime(e)
    i=datetime.timedelta(seconds=interval)
    l=[]

```

```

while st<=et:
    l.append(st)
    st+=i
return l
def merge(scale,data):
    l=[]
    while len(scale)!=0 and len(data)!=0:
        s,d=scale.pop(0),data.pop(0)
        if s<=d[0]:
            l.append(s)
            data.insert(0,d)
        else:
            l.append(d)
            scale.insert(0,s)
    if len(scale)==0:
        for i in data:
            l.append(i)
    elif len(data)==0:
        for i in scale:
            l.append(i)
    return l
def power_sum(l):
    p=[]
    sum=0
    for i in l:
        if type(i)==type():#if tuple
            sum+=i[1]*10#10secs
        else:
            p.append((i,sum))
            sum=0
    return p
def view(connection,channel,st,et,interval,raw):
    data=ch(connection,channel,st,et)
    if raw=='true':
        return data
    elif raw=='false':
        scale=scale_generate(st,et,interval)
        l=merge(scale,data)
        return power_sum(l)

```

power_single.py

```

#!/usr/bin/python

#params
hostname="localhost"
username="*****"
pswd="*****"
dbn="powerdb"
#script db connection

```

```

import ops_db
import ops_table
connection=ops_db.connect(hostname,username,pswd)
ops_db.select(connection,dbn)
#script get params
import cgi
form=cgi.FieldStorage()
print "Content-type: text/html"
print
print "<html>"
print "</html>"
ch=form["ch"].value
st,et=form["st"].value.replace('s','')
.replace('d','-').replace('c',''),form["et"].value.replace('s','').replace('d','-').replace('c','')
interval=int(form["interval"].value)
raw=form["raw"].value
#script make inquiries
import inquiry
l=inquiry.view(connection,ch,st,et,interval,raw)
if raw=='true':
    for i in l:
        print "<p>"
        print i[0],i[1],"Watts"
        print "</p>"
elif raw=='false':
    for i in l:
        print "<p>"
        print i[0],i[1],"J"
        print "</p>"

```

practical_visualizer.py

```

#config & read file
fn="20110520.txt"
mode="humidity"
interval=3600

st=fn[0:4]+"-"+fn[4:6]+"-"+fn[6:8]+" 00:00:00"
#####SCRIPT STARTS FROM HERE#####
import matplotlib.pyplot as plt
import datetime
import os

#make data list
st_d=datetime.datetime.strptime(st,"%Y-%m-%d %H:%M:%S")
et_d=st_d+datetime.timedelta(days=1)
d=datetime.timedelta(seconds=1)
c_t=st_d
data={}

```

```

while c_t<=et_d:
    data[c_t]=0
    c_t+=d
#read data file
for l in open(fn,'r').readlines():
    l=l.strip()
    if l=="": continue
    dt=l.split(" ")[0]+" "+l.split(" ")[1]
    data[datetime.datetime.strptime(dt,"%Y-%m-%d %H:%M:%S")]=float(l.split(" ")[2])
#make data
data1=[]
for i in data.keys():
    data1.append([i,data[i]])
data1.sort()
data2=[]
for i in data1:
    data2.append(i[1])
#discrete/process data: interpolation between two data
#plot
fig=plt.figure()
ax=fig.add_subplot(111)
if mode=="power":
    ax.plot(range(len(data2)),data2)
    ax.set_xlabel("Time:s")
    ax.set_ylabel("Power:Watts")
elif mode=="humidity":
    ax.plot(range(len(data2)),data2)
    ax.set_xlabel("Time:s")
    ax.set_ylabel("HUmidity:Percents")
elif mode=="energy":
    data3=[]
    while len(data2)!=0:
        sum=0
        for i in range(interval):
            if len(data2)!=0:
                sum+=data2.pop(0)
        sum=sum/3600000.
        data3.append(sum)
    ax.bar(range(len(data3)),data3,alpha=0.5)
    ax.set_xlabel("Time:"+str(interval)+"s")
    ax.set_ylabel("Energy:kwh")
ax.grid(True)
plt.show()

```

Part B: Source code of SE

appliance.py

```
#####PARAMS READER#####
#common
def config_file_read(fn):
    d={}
    for line in [i.strip() for i in open(fn,'r').readlines()]:
        if line=="#":continue#take out
        l=line.split("=")
        if l[0]=="node_table":
            d["node_table"]=node_table_read(l[1])
        elif l[0]=="transition_chart":
            d["transition_chart"]=transition_chart_read(l[1])
        elif l[0]=="route_table":
            d["route_table"]=route_table_read(l[1])
        elif l[0]=="pcf_params":
            d["pcf_params"]=pcf_params_read(l[1])
    return d
def node_table_read(str):
    dict={}
    for i in str.split(";"):
        dict[i.split(":")[0]]=i.split(":")[1],0]
    return dict
def transition_chart_read(str):
    dict={}
    for i in str.split(";"):
        dict[i.split(":")[0]]=i.split(":")[1],0,0]
    return dict
def route_table_read(str):#des_node,cur_node,nex_node,wait_limit;
    dict={}
    for i in str.split(";"):
        dict[i.split(":")[0]]=i.split(":")[1],0]
    return dict
def pcf_params_read(str):
    dict={}
    for i in str.split(";"):
        t=[]
        for j in i.split(":")[1].split("/"):
            t.append(j)
        dict[i.split(":")[0]]=t
    return dict
#####PCF & RANDOM FUNCS#####
import random
import math
def distortor(base,r):#non-negative
    result=base*(1+random.uniform(-r,r))
```

```

    if result<0: return 0
    else: return result
def pcf(p,cur_t,glb_t):
    func,distortion,abnormal=p
    func_type,func_p=func.split("|")
    #####FUNC TYPES###
    if func_type=='0':#func type 0
        val=float(func_p)
    elif func_type=='1':#func type 1
        p1,p2,p3=func_p.split(",")
        p1,p2,p3=float(p1),float(p2),float(p3)
        val=p1+p2*math.exp(p3*cur_t)
    elif func_type=='2':
        p1,p2,p3,p4=func_p.split(",")
        p1,p2,p3,p4=float(p1),float(p2),float(p3),float(p4)
        val=p1+p2*math.exp((glb_t-p3)*(p3-glb_t)/p4/p4)
    #####DISTORTION###
    if "," in distortion:
        dis_r,dis_p=distortion.split(",")
        dis_r,dis_p=float(dis_r),float(dis_p)
        if random.uniform(0,1)<float(dis_p):
            val=val*(1+random.uniform(-dis_r,dis_r))
    else:
        dis_r=float(distortion)
        val=val*(1+random.uniform(-dis_r,dis_r))
    #####ABNORMAL###
    abnormal_type,abnormal_p=abnormal.split("|")
    ab_base,ab_r,ab_prob=abnormal_p.split(",")
    if random.uniform(0,1)<float(ab_prob):
        ab_base,ab_r=float(ab_base),float(ab_r)
        ab=ab_base*(1+random.uniform(-ab_r,ab_r))
        if abnormal_type=="+" and ab>val:
            return int(ab)
        elif abnormal_type=="-" and ab<val:
            return int(ab)
        elif abnormal_type=="?":
            return int(ab)
    return int(val)
#####UTILITY#####
import os
def s2d(s):
    d={}
    for item in s.split(","):#sep1
        key,val=item.split("=")#sep2
        d[key]=val
    return d
#####CLASS APPLIANCE#####
class appliance:
    def __init__(self,config_fn,init_stat):
        self.cur_stat=init_stat

```



```

self.cur_timer=0
self.global_timer=0
#self.internal_stat="idling"
self.cls=config_fn.split(os.sep).pop().split(".")[2].split("|")[1]
self.pwr=0
self.req_lst=[]
self.req="n/a"
self.params=config_file_read(config_fn)
#init node_table
for i in self.params["node_table"].keys():
    if self.params["node_table"][i][0]=="n/a":continue
    t=self.params["node_table"][i][0].split(",")
    self.params["node_table"][i][1]=distortor(float(t[0]),float(t[1]))
#init transition_chart
for i in self.params["transition_chart"].keys():
    if self.params["transition_chart"][i][0]=="n/a":continue###5.26
    t=self.params["transition_chart"][i][0].split(",")#re-calculate a wl in
transition_chart
    self.params["transition_chart"][i][1]=distortor(float(t[0]),float(t[1]))
#init route_table if be
if self.params.has_key("route_table"):
    for i in self.params["route_table"].keys():
        if self.params["route_table"][i][0]=="n/a":continue
        t=self.params["route_table"][i][0].split(",")#re-calculate a wl in
route_table
        self.params["route_table"][i][1]=distortor(float(t[0]),float(t[1]))

def run(self):
    #output pwr, if abnormal is greater, output ab_pwr
    p=self.params["pcf_params"][self.cur_stat]
    self.pwr=pcf(p,self.cur_timer,self.global_timer)
    #timer increase
    self.cur_timer+=1
    self.global_timer+=1
    for i in self.params["transition_chart"].keys():
        if self.params["transition_chart"][i][0]=="n/a":continue###5.26
        self.params["transition_chart"][i][2]+=1
    #get req if none
    if self.req=="n/a":
        self.req=self.get_req()
    #####SWITCH#####
    #behave according to transition chart only
    if self.req=="n/a":
        #condition, no switch
        if self.params["node_table"][self.cur_stat][0]=="n/a" or
self.cur_timer<self.params["node_table"][self.cur_stat][1]: return
        #find the next node of the highest period which is eligible to be switched to
        key=""
        for k in self.params["transition_chart"].keys():
            if k.split(",")[0]!=self.cur_stat: continue

```

```

        if self.params["transition_chart"][k][0]=="n/a": continue###5.26
        if
float(self.params["transition_chart"][k][1])>float(self.params["transition_chart"][k][2]):
continue
        if key=="": key=k
        elif
float(self.params["transition_chart"][key][1])<float(self.params["transition_chart"][k][
1]): key=k
        else: pass
        #action
        if key=="": return
        #reset timers
        self.cur_timer=0
        self.params["transition_chart"][key][2]=0
        #change wait limit of old cur_stat in node_table
        node_p=self.params["node_table"][self.cur_stat][0].split(",")

self.params["node_table"][self.cur_stat][1]=distortor(float(node_p[0]),float(node_p[
1]))

        #change wait limit in transition_chart
        t=self.params["transition_chart"][key][0].split(",")#re-calculate a wl in
transition_chart
        self.params["transition_chart"][key][1]=distortor(float(t[0]),float(t[1]))
        #change cur_stat
        self.cur_stat=key.split(",")[1]
        #behave according to route table and transition chart
        elif self.req!="n/a":
            d=s2d(self.req)
            #target_mod,duration=d["TARGET_MOD"],d["DURATION"]###5.26
            #condition
            #if not desired mode, in routing
            if self.cur_stat!=d["TARGET_MOD"]:
                #find the transition(key) in route table
                key=""
                for i in self.params["route_table"].keys():
                    l=i.split(",")
                    #print i,"====",d["TARGET_MOD"],self.cur_stat
                    if l[0]==d["TARGET_MOD"] and l[1]==self.cur_stat:
                        key=i
                #print key
                #find wait limit
                wl=0
                if self.params["route_table"][key][0]=="n/a":#according to the node table
                    wl=self.params["node_table"][self.cur_stat][1]
                    #re-calculate a wl in node_table
                    t=self.params["node_table"][self.cur_stat][0].split(",")

self.params["node_table"][self.cur_stat][1]=distortor(float(t[0]),float(t[1]))
                elif self.params["route_table"][key][0!="n/a":#according to the route
table

```

```

        wl=self.params["route_table"][key][1]
        #re-calculate a wl in route_table
        t=self.params["route_table"][key][0].split(",")
        self.params["route_table"][key][1]=distortor(float(t[0]),float(t[1]))
#switch, if cur_timer exceeds wait limit
if self.cur_timer>=wl:
    #reset timers
    self.cur_timer=0

    self.params["transition_chart"][self.cur_stat+","+key.split(",")[2]][2]=0
    #change wait limit in transition_chart if it be
    if
self.params["transition_chart"][self.cur_stat+","+key.split(",")[2]][0]!="n/a":

        t=self.params["transition_chart"][self.cur_stat+","+key.split(",")[2]][0].split(",")#re-c
        alculate a wl in transition_chart

        self.params["transition_chart"][self.cur_stat+","+key.split(",")[2]][1]=distortor(float(
        t[0]),float(t[1]))

            #change cur_stat
            self.cur_stat=key.split(",")[2]
#condition
#if desired mode, in working
elif self.cur_stat==d["TARGET_MOD"]:
    #find wait limit
    wl=0
    if d["DURATION"]=="n/a":#according to node table
        wl=self.params["node_table"][self.cur_stat][1]
        #re-calculate a wl in node_table
        t=self.params["node_table"][self.cur_stat][0].split(",")

        self.params["node_table"][self.cur_stat][1]=distortor(float(t[0]),float(t[1]))
        #select longer wl if requested duration is not n/a
        elif d["DURATION"]!="n/a" and
self.params["node_table"][self.cur_stat][0]=="n/a":
            wl=int(d["DURATION"])
        elif d["DURATION"]!="n/a" and
self.params["node_table"][self.cur_stat][0]!="n/a":
            if
int(d["DURATION"])>=self.params["node_table"][self.cur_stat][1]:
                wl=int(d["DURATION"])
            elif
int(d["DURATION"])<self.params["node_table"][self.cur_stat][1]:
                wl=self.params["node_table"][self.cur_stat][1]

        t=self.params["node_table"][self.cur_stat][0].split(",")#re-calculate a wl in
        node_table

        self.params["node_table"][self.cur_stat][1]=distortor(float(t[0]),float(t[1]))
        #switch, if exceeds wait limit

```

```

if self.cur_timer>=wl:
    #find next req
    if len(self.req_lst)==0:
        self.req="n/a"
    else:
        self.req=self.req_lst[0]
    #if no next req
    if self.req=="n/a":
        #fall back according to transition chart
        #find the next node of the highest period which is eligible to be
switched to
        key=""
        for k in self.params["transition_chart"].keys():
            if k.split(",")[0]!=self.cur_stat: continue
            if self.params["transition_chart"][k][0]=="n/a":
continue###5.26 never goes to where it would never go
            if
float(self.params["transition_chart"][k][2])<float(self.params["transition_chart"][k][1]):
continue
                if key=="": key=k
                elif
float(self.params["transition_chart"][key][1])<float(self.params["transition_chart"][k][
1]): key=k
                    else: pass
        #action
        if key=="": return
        #reset timers
        self.cur_timer=0
        self.params["transition_chart"][key][2]=0
        #change wait limit in transition_chart
        t=self.params["transition_chart"][key][0].split(",")#re-calculate
a wl in transition_chart

        self.params["transition_chart"][key][1]=distortor(float(t[0]),float(t[1]))
        #change cur_stat
        self.cur_stat=key.split(",")[1]
        self.req="n/a"
        #if next req be
        elif self.req!="n/a":
            #check target_mod
            d=s2d(self.req)
            #if next target_mod different, fall back according to transition
chart
            if d["TARGET_MOD"]!=self.cur_stat:
                #fall back according to transition chart
                #find the next node of the highest period which is eligible to
be switched to
                key=""
                for k in self.params["transition_chart"].keys():
                    if k.split(",")[0]!=self.cur_stat: continue

```

```

        if self.params["transition_chart"][k][0]=="n/a":
continue###5.26 never goes to where it would never go
        if
float(self.params["transition_chart"][k][2])<float(self.params["transition_chart"][k][1]):
continue
        if key=="": key=k
        elif
float(self.params["transition_chart"][key][1])<float(self.params["transition_chart"][k][
1]): key=k
        else: pass
        #action
        if key=="": return
        #reset timers
        self.cur_timer=0
        self.params["transition_chart"][key][2]=0
        #change wait limit in transition_chart

        t=self.params["transition_chart"][key][0].split(",")#re-calculate a wl in
transition_chart

        self.params["transition_chart"][key][1]=distortor(float(t[0]),float(t[1]))
        #change cur_stat
        self.cur_stat=key.split(",")[1]
        self.req="n/a"
        #if next target_mod same, serve
        elif d["TARGET_MOD"]==self.cur_stat:
        #action
        #reset timer
        self.req=self.get_req()
        self.cur_timer=0

def rcv(self,req):
    #shared
    if self.cls=="shared":
        if self.req=="n/a":
            self.req_lst.append(req)
        elif self.req!="n/a":
            cur_d=s2d(self.req)
            req_d=s2d(req)
            if req_d["TARGET_MOD"]==cur_d["TARGET_MOD"] and
int(req_d["DURATION"])>=(int(cur_d["DURATION"])-self.cur_timer):
                #reset duration

        save=self.req.split("DURATION=")[0]+"DURATION="+req_d["DURATION"]
        self.req=save
        #reset timer
        self.cur_timer=0

    #mutex
    if self.cls=="mutex":
        self.req_lst.append(req)

```

```

    #personal
    if self.cls=="personal":
        self.req_lst.append(req)
def get_req(self):#####SCHEDULING HERE
    if len(self.req_lst)==0:
        return "n/a"
    else:
        req=self.req_lst.pop(0)
        return req
def new_day(self):
    self.global_timer=0

```

agent.py

```

import random
import numpy as np
import os

def nd(mu,sigma):
    X=np.random.normal(mu,sigma)
    return int(X)

def params2val(p):
    mu,sigma=[int(i) for i in p.split("/")]#sep3
    return nd(mu,sigma)

def read_profile(fn):
    d={}
    lines=[i.strip() for i in open(fn).readlines()]
    for line in lines:
        if line==" " or line[0]=="#":continue#take out empty or comment line
        dict={}
        for item in line.split(","):#sep1
            key,val=item.split("=")#sep2
            dict[key]=val
        #runtime parameters
        dict["TIMER"]=0
        dict["COUNTER"]=0#maxium run count, int
        dict["STATUS"]="waiting"
        #start options
        if dict.has_key("START_PARAMS"):
            dict["START"]=params2val(dict["START_PARAMS"])#calculate a new start,
int
        if dict.has_key("PROBABILITY"):
            dict["PROBABILITY"]=float(dict["PROBABILITY"])#probability, float
        #end options
        if dict.has_key("END_PARAMS"):

            dict["DURATION"]=params2val(dict["END_PARAMS"])-dict["START"]#calculate a
new end, then duration, int

```

```

        if dict.has_key("DURATION_PARAMS") and
dict["DURATION_PARAMS"]!="n/a":
        dict["DURATION"]=params2val(dict["DURATION_PARAMS"])#calculate a
new duration, int
        d[dict["PID"]]=dict
        return d

def is_blocked(group,d):#test whether the group is blocked
    for i in d.keys():
        if d[i]["GROUP"]==group and d[i]["STATUS"]=="running":
            return True
    return False

def in_prob(p):#test whether to start if prob is given
    if random.random()<p:
        return True
    else:
        return False

def req2d(r):
    d={}
    for i in r.split(","):
        key,val=i.split("=")
        d[key]=val
    return d

class agent:
    def __init__(self,fn):
        self.fn=fn
        self.t=0
        self.p=read_profile(fn)
        self.rdy_lst=[]
        self.req_log=[]

    def act(self):
        self.t+=1
        for k in self.p.keys():
            if self.p[k]["STATUS"]=="waiting":
                change_flag=False
                #condition 1
                if self.p[k]["PPID"]=="n/a" or
self.p[self.p[k]["PPID"]]["STATUS"]=="running":pass
                else:continue
                #condition 2
                if self.p[k]["COUNTER"]<int(self.p[k]["MULTI"]):pass
                else:continue
                #condition 3
                if self.p[k].has_key("START"):
                    if self.p[k]["PPID"]=="n/a":
                        t=self.t

```

```

else:
    t=self.p[self.p[k]["PPID"]]["TIMER"]
    if t>self.p[k]["START"]:#if parent process TIMER exceeds START
        change_flag=True
elif self.p[k].has_key("PROBABILITY"):
    if in_prob(float(self.p[k]["PROBABILITY"])):
        change_flag=True
#action:waiting-->running/ready
if change_flag==True:
    #reset timer
    self.p[k]["TIMER"]=0
    #change status
    self.p[k]["COUNTER"]+=1
    #re-calculate start if be
    if self.p[k].has_key("START"):
        self.p[k]["START"]=params2val(self.p[k]["START_PARAMS"])
    #decides destination: running/ready
    if is_blocked(self.p[k]["GROUP"],self.p)==False:
        self.p[k]["STATUS"]="running"
        #####SEND REQUEST HERE#####
        self.send_req(k)
    elif is_blocked(self.p[k]["GROUP"],self.p)==True:
        self.p[k]["STATUS"]="ready"
        self.rdy_lst.append(self.p[k]["PID"])
        #####READY INFO#####
        #print "PID ",self.p[k]["PID"]," READY AT ",self.t
elif self.p[k]["STATUS"]=="running":
    self.p[k]["TIMER"]+=1#running timer
    change_flag=False
    #condition 1
    if self.p[k]["PPID"]=="n/a":pass
    elif self.p[self.p[k]["PPID"]]["STATUS"]!="running":
        change_flag=True
    #condition 2
    if self.p[k].has_key("DURATION")==True and
self.p[k]["TIMER"]>self.p[k]["DURATION"]:
        change_flag=True
    #action:running-->waiting
    if change_flag==True:
        #reset timer
        self.p[k]["TIMER"]=0
        #change status
        self.p[k]["STATUS"]="waiting"
        #re-calculate duration
        if self.p[k].has_key("END_PARAMS"):

            self.p[k]["DURATION"]=params2val(self.p[k]["END_PARAMS"])-self.p[k]["START"]
            #calculate a new end, then duration, int
            if self.p[k].has_key("DURATION_PARAMS") and
self.p[k]["DURATION_PARAMS"]!="n/a":

```



```

        self.p[k]["DURATION"]=params2val(self.p[k]["DURATION_PARAMS"])#calculate
a new duration, int
        for kk in self.p.keys():
            if self.p[k]["PID"]==self.p[kk]["PPID"]:
                self.p[kk]["COUNTER"]=0
            #####WRITE DURATION INFO WHEN END#####
        for i in range(len(self.req_log)):
            req=req2d(self.req_log[i])
            if req["PID"]==self.p[k]["PID"] and req["DURATION"]=="n/a":
                self.chg_dur(i)
            #print "PID ",self.p[k]["PID"]," FINISHED AT ",self.t
if len(self.rdy_lst)!=0:
    for i in range(len(self.rdy_lst)):
        pid=self.rdy_lst[i]
        #condition 1
        if self.p[pid]["PPID"]=="n/a":pass
        elif self.p[self.p[pid]["PPID"]]["STATUS"]!="running":
            self.p[pid]["TIMER"]=0
            self.p[pid]["STATUS"]="waiting"
            self.rdy_lst[i]="deleted"
            #####CANCELED INFO#####
            #print "PID ",self.p[pid]["PID"]," CANCELED AT ",self.t
            continue
        else:pass
        #condition 2
        if is_blocked(self.p[pid]["GROUP"],self.p)==True:pass
        else:
            self.p[pid]["TIMER"]=0
            self.p[pid]["STATUS"]="running"
            self.rdy_lst[i]="deleted"
            #####SEND REQUEST HERE#####
            self.send_req(k)
            #print "PID ",self.p[pid]["PID"]," READY-->STARTED AT ",self.t
            continue
    #clean up rdy_lst
    l=[]
    for i in self.rdy_lst:
        if i!="deleted":
            l.append(i)
    self.rdy_lst=l

def restart(self):
    self.t=0
    self.p=read_profile(self.fn)
    self.rdy_lst=[]
    self.req_log=[]

def send_req(self,k):
    """

```

```

        if self.p[k]["TYPE"]=="a":
            print "TIME ",self.t," ",
            print "AGENT ",self.fn.split(".")[1]+"."+self.fn.split(".")[2]," ",
            print "TARGET_APP ",self.p[k]["TARGET_APP"]," ",
            print "TARGET_MOD ",self.p[k]["TARGET_MOD"]," ",
            if self.p[k].has_key("DURATION"):
                print "DURATION ",self.p[k]["DURATION"]," ",
            else:
                print "DURATION ","n/a"," ",
            print "PRIORITY",self.p[k]["PRIORITY"]
        """
        if self.p[k]["TYPE"]=="a":
            time="TIME="+str(self.t)+","

        aid="AGENT="+self.fn.split(".")[1]+"."+self.fn.split(".")[2]+"."+self.fn.split(".")[3]+"."
+self.fn.split(".")[4]+","
        pid="PID="+self.p[k]["PID"]+", "
        app="TARGET_APP="+self.p[k]["TARGET_APP"]+", "
        mod="TARGET_MOD="+self.p[k]["TARGET_MOD"]+", "
        if self.p[k].has_key("DURATION"):
            dur="DURATION="+str(self.p[k]["DURATION"])+","
        else:
            dur="DURATION=n/a,"
        pri="PRIORITY="+self.p[k]["PRIORITY"]
        req=time+aid+pid+app+mod+dur+pri
        #print req
        self.req_log.append(req)

    def chg_dur(self,i):
        req=req2d(self.req_log[i])
        time="TIME="+req["TIME"]+", "
        aid="AGENT="+req["AGENT"]+", "
        pid="PID="+req["PID"]+", "
        app="TARGET_APP="+req["TARGET_APP"]+", "
        mod="TARGET_MOD="+req["TARGET_MOD"]+", "
        dur="DURATION="+str(self.t-int(req["TIME"]))+","
        pri="PRIORITY="+req["PRIORITY"]
        req_s=time+aid+pid+app+mod+dur+pri
        self.req_log.pop(i)
        self.req_log.insert(i,req_s)

```

init.py

```

import random
import os

def quantity(s):#get quantities of agents and appliances in a cell
    l=[]
    for i in s.split(" | "):
        val=int(i.split(":")[0])

```

```

        count=int(float(i.split(":")[1])*100)
        for i in range(count):
            l.append(val)
        return random.sample(l,1)[0]

def f2l_d(fn):
    l=[]
    lines=[i.strip() for i in open(fn,'r').readlines()]
    for line in lines:
        if line==" or line[0]=="#":continue#take out empty or comment line
        if line.split("=")[0]=="DAYS":continue#ignore key "DAYS"
        d={}
        for item in line.split(","):#sep1
            key,val=item.split("=")#sep2
            d[key]=val
        l.append(d)
    return l

#####SCRIPT STARTS HERE#####
#make cell name list
cells=[]
for i in f2l_d("init.cfg"):
    count=int(i["COUNT"])
    for ii in range(count):
        cells.append(i["TYPE"]+"."+str(ii+1))

#make agent and appliance name list for each cell in cells
agents=[]
appliances=[]
prefix=os.getcwd()+os.sep+"templates_cell"+os.sep+"cell."#path & fn format
for cell in cells:
    entities=f2l_d(prefix+cell.split(".")[0])
    for entity in entities:
        count=quantity(entity["COUNT"])
        if entity.has_key("MEMBER"):
            for i in range(count):
                agents.append("agent."+entity["MEMBER"]+"."+str(i+1)+"."+cell)
        if entity.has_key("APPLIANCE"):
            for i in range(count):
                appliances.append("appliance."+entity["APPLIANCE"]+"."+entity["TYPE"]+"."+str(i+1)+"."+cell)

#make list of cells: [{cell_name:str,member:[str,...],appliance:[str,...]}...]
cells2=[]
for i in cells:
    d={}
    d["CELL"]=i
    d["MEMBER"]=[]
    d["APPLIANCE"]=[]

```

```

for a in agents:
    t=a.split(".")[2:]
    if i==t[0]+"."+t[1]:
        d["MEMBER"].append(a)
for a in appliances:
    t=a.split(".")[2:]
    if i==t[0]+"."+t[1]:
        d["APPLIANCE"].append(a)
cells2.append(d)

#write each cell's cfg into file
f=open("cells.cfg",'w')#output fn
for i in cells2:
    s="CELL="+i["CELL"]+", "
    for ii in i["MEMBER"]:
        s=s+"MEMBER="+ii+", "
    for ii in i["APPLIANCE"]:
        s=s+"APPLIANCE="+ii+", "
    s=s[:-1]
    f.write(s)
    f.write("\n")
f.close()

#end message
print "Cells configuration generated"

```

profile_gen.py

```

import random
import numpy as np
import os

#random func normal distr.
def nd(mu,sigma):
    if sigma==0:
        return int(mu)
    X=np.random.normal(mu,sigma)
    return int(X)
def params_nd(base,var):
    mu1,mu2=[float(i) for i in base.split("/")]#sep3
    rt1,rt2=[float(i) for i in var.split("/")]#sep3
    return str(nd(mu1,mu1*rt1))+"/"+str(nd(mu2,mu2*rt2))
def params_nd1(base,var):
    mu1,mu2=[float(i) for i in base.split(",")]#sep3
    rt1,rt2=[float(i) for i in var.split(",")]#sep3
    return str(nd(mu1,mu1*rt1))+","+str(nd(mu2,mu2*rt2))
def params_nd2(base,var):
    base,var=float(base),float(var)
    return str(nd(base,base*var))
#random func2 uniform distr.

```

```

def uniform(base,r):
    return int(base*(1+random.uniform(-r,r)))
def params_uniform(base,r):
    base,r=float(base),float(r)
    return str(uniform(base,r))

def pcf_change(str0,str1):
    #base
    base0,dist0,abn0=str0.split("/")
    base_l0=base0.split("|")[1].split(",")
    dist_l0=dist0.split(",")
    abn_l0=abn0.split("|")[1].split(",")
    #var
    base1,dist1,abn1=str1.split("/")
    base_l1=base1.split("|")[1].split(",")
    dist_l1=dist1.split(",")
    abn_l1=abn1.split("|")[1].split(",")
    #output
    base_l,dist_l,abn_l=[],[],[]
    #get len
    for i in range(len(base_l0)):
        base_l.append(params_nd2(base_l0[i],base_l1[i]))
    for i in range(len(dist_l0)):
        dist_l.append(params_nd2(dist_l0[i],dist_l1[i]))
    for i in range(len(abn_l0)):
        abn_l.append(params_nd2(abn_l0[i],abn_l1[i]))
    #result str
    #pcf
    s=""
    s+=base0.split("|")[0]+"| "
    for i in base_l:
        s+=i+", "
    s=s[:-1]
    #dist
    s+="/"
    for i in dist_l:
        s+=i+", "
    s=s[:-1]
    #abn
    s+="/"
    s+=abn0.split("|")[0]+"| "
    for i in abn_l:
        s+=i+", "
    s=s[:-1]
    return s

def d2s(d):
    s=""
    for k in d.keys():
        s+=k

```

```

    s+=":"
    s+=d[k]
    s+=";"
    return s[:-1]

```

```

def param_prob(base,var):
    mu=float(base)
    sigma=mu*float(var)
    X=np.random.normal(mu,sigma)
    return str(X)

```

#delete all files in dir(under current dir)

```

def remove_files(dn):
    #os dependent
    s=os.sep
    path=os.getcwd()+s+dn+s
    for i in os.listdir(path):
        os.remove(path+i)

```

def agent_profile_gen(std_fn,cfg_fn,output_fn):

```

    #os dependent
    s=os.sep
    path1=os.getcwd()+s+"templates_agent"+s#input files path
    path2=os.getcwd()+s+"profiles_agent"+s#output files path
    #std lst of dicts
    std_l=[]
    lines=[i.strip() for i in open(path1+std_fn).readlines()]
    for line in lines:
        if line==" " or line[0]=="#":continue#take out empty or comment line
        dict={}
        for item in line.split(","):#sep1
            key,val=item.split("=")#sep2
            dict[key]=val
        std_l.append(dict)
    #cfg lst of dicts
    cfg_l=[]
    lines=[i.strip() for i in open(path1+cfg_fn).readlines()]
    for line in lines:
        if line==" " or line[0]=="#":continue#take out empty or comment line
        dict={}
        for item in line.split(","):#sep1
            key,val=item.split("=")#sep2
            dict[key]=val
        cfg_l.append(dict)
    #make changes
    output=[]
    for i in std_l:
        temp=i
        for ii in cfg_l:
            if i["PID"]==ii["PID"]:

```

```

        if ii.has_key("START_PARAMS"):

temp["START_PARAMS"]=params_nd(i["START_PARAMS"],ii["START_PARAMS"
])
        if ii.has_key("PROBABILITY"):

temp["PROBABILITY"]=param_prob(i["PROBABILITY"],ii["PROBABILITY"])
        if ii.has_key("END_PARAMS"):

temp["END_PARAMS"]=params_nd(i["END_PARAMS"],ii["END_PARAMS"])
        if ii.has_key("DURATION_PARAMS"):

temp["DURATION_PARAMS"]=params_nd(i["DURATION_PARAMS"],ii["DURATI
ON_PARAMS"])
        output.append(temp)
#write file
#cls=std_fn.split(".")[0]
f=open(path2+output_fn,"w")
for i in output:
    f.write("PID="+i["PID"]+",")
    f.write("PPID="+i["PPID"]+",")
    f.write("GROUP="+i["GROUP"]+",")
    f.write("TYPE="+i["TYPE"]+",")
    if i["TYPE"]=="a":
        f.write("MULTI="+i["MULTI"]+",")
        f.write("TARGET_APP="+i["TARGET_APP"]+",")
        f.write("TARGET_MOD="+i["TARGET_MOD"]+",")
        f.write("PRIORITY="+i["PRIORITY"]+",")
    elif i["TYPE"]=="s":
        f.write("MULTI="+i["MULTI"]+",")
    if i.has_key("START_PARAMS"):
        f.write("START_PARAMS="+i["START_PARAMS"]+",")
    if i.has_key("PROBABILITY"):
        f.write("PROBABILITY="+i["PROBABILITY"]+",")
    if i.has_key("END_PARAMS"):
        f.write("END_PARAMS="+i["END_PARAMS"])
    if i.has_key("DURATION_PARAMS"):
        f.write("DURATION_PARAMS="+i["DURATION_PARAMS"])
    f.write("\n")
f.close()

def appliance_profile_gen(std_fn,cfg_fn,output_fn):
    #os dependent
    s=os.sep
    path1=os.getcwd()+s+"templates_appliance"+s#input files path
    path2=os.getcwd()+s+"profiles_appliance"+s#output files path
    #std lst of dicts
    std_d={}
    lines=[i.strip() for i in open(path1+std_fn).readlines()]
    for line in lines:

```

```

        if line==" or line[0]=="#":continue#take out empty or comment line
        d={}
        for item in line.split("=")[1].split(";"):
            key,val=item.split(":")
            d[key]=val
        std_d[line.split("=")[0]]=d
#cfg lst of dicts
cfg_d={}
lines=[i.strip() for i in open(path1+cfg_fn).readlines()]
for line in lines:
    if line==" or line[0]=="#":continue#take out empty or comment line
    d={}
    for item in line.split("=")[1].split(";"):
        key,val=item.split(":")
        d[key]=val
    cfg_d[line.split("=")[0]]=d
#print std_d
#print cfg_d
#make changes
output={}
for k in std_d.keys():
    output[k]=std_d[k]
    if cfg_d.has_key(k)==False: continue#continue if no such a key
    if k!="pcf_params":
        for kk in cfg_d[k].keys():
            output[k][kk]=params_nd1(output[k][kk],cfg_d[k][kk])
    elif k=="pcf_params":
        for kk in cfg_d[k].keys():
            output[k][kk]=pcf_change(std_d[k][kk],cfg_d[k][kk])
#write file
#?cls=std_fn.split(".")[0]
f=open(path2+output_fn,"w")
f.write("node_table=")
f.write(d2s(output["node_table"]))
f.write("\n")
f.write("transition_chart=")
f.write(d2s(output["transition_chart"]))
f.write("\n")
if output.has_key("route_table"):
    f.write("route_table=")
    f.write(d2s(output["route_table"]))
    f.write("\n")
f.write("pcf_params=")
f.write(d2s(output["pcf_params"]))
f.write("\n")
f.close()

def f2l_l(fn):
    l=[]
    lines=[i.strip() for i in open(fn,'r').readlines()]

```



```

    for line in lines:
        if line==" or line[0]=="#":continue#take out empty or comment line
        #if line.split("=")[0]=="CELL":continue#ignore key "CELL"
        ll=[]
        for item in line.split(","):#sep1
            ll.append(item)
        l.append(ll)
    return l
#appliance_profile_gen("appliance.microwave_oven","appliance.microwave_oven.cfg",
123")

#####SCRIPT STARTS HERE#####
#read cfg
agents=[]
appliances=[]
for items in f2l_1("cells.cfg"):
    for item in items:
        if item.split("=")[0]=="MEMBER":
            agents.append(item.split("=")[1])
        if item.split("=")[0]=="APPLIANCE":
            appliances.append(item.split("=")[1])
#make agent profile
remove_files("profiles_agent")
for a in agents:
    t=a.split(".")
    input_fn=t[0]+"."+t[1]
    agent_profile_gen(input_fn,input_fn+".cfg",a)
print "Agent profiles generated"
#make appliance profile
remove_files("profiles_appliance")
for a in appliances:
    t=a.split(".")
    input_fn=t[0]+"."+t[1]
    appliance_profile_gen(input_fn,input_fn+".cfg",a)
#end message
print "Appliance profiles generated"

```

req_gen.py

```

import agent
import os

def params_in_comment(fn):
    d={}
    for line in [i.strip() for i in open(fn,'r').readlines()]:
        if line[0]=="#":
            l=line[1:].split("=")
            d[l[0]]=l[1]
    return d

```

```

def remove_files(dn):
    #os dependent
    s=os.sep
    path=os.getcwd()+s+dn+s
    for i in os.listdir(path):
        os.remove(path+i)

#input
s=os.sep
for i in open("init.cfg",'r').readlines():
    if i.split("=")[0]=="DAYS":
        days=int(i.split("=")[1])
path1=os.getcwd()+s+"profiles_agent"
path2=os.getcwd()+s+"logs_req"
profiles=os.listdir(path1)
#actions
remove_files("logs_req")
#create req_logs
print "Generating logs"
for day in range(1,days+1):
    for profile in profiles:
        print "Generating req_log for "+profile+" at day "+str(day)
        a=agent.agent(path1+s+profile)
        for sec in range(1,86401):
            a.act()
            f=open(path2+s+profile.split(".")[3]+". "+profile.split(".")[4]+".day."+str(day),'a+')
            for req in a.req_log:
                f.write(req)
                f.write("\n")

```

world.py

```

import os
import random
import appliance

def re_time(s,d):
    l=s.split(",AGENT")
    ll=l[0].split("=")
    new_s="TIME="+str(d*86400+int(ll[1]))+",AGENT"+l[1]
    return new_s

def t(s):
    l=s.split(",AGENT")
    ll=l[0].split("=")
    t=int(ll[1])
    return t

def sort_log(l):#insert sort
    for i in range(1, len(l)):

```

```

    save=l[i]
    j=i
    while j>0 and t(l[j-1])>t(save):
        l[j]=l[j - 1]
        j-=1
    l[j]=save
    return l

```

```

def remove_files(dn):
    #os dependent
    s=os.sep
    path=os.getcwd()+s+dn+s
    for i in os.listdir(path):
        os.remove(path+i)

```

```

def req_handler(cell,req,t):
    #read req
    agent,target_app,target_mod,duration="", "", "", ""
    items=req.split(",")
    for item in items:
        if item.split("=")[0]=="AGENT":
            agent=item.split("=")[1]
        elif item.split("=")[0]=="TARGET_APP":
            target_app=item.split("=")[1]
        elif item.split("=")[0]=="TARGET_MOD":
            target_mod=item.split("=")[1]
        elif item.split("=")[0]=="DURATION":
            duration=item.split("=")[1]
    agent_type=agent.split(".")[0]
    agent_n=agent.split(".")[1]
    rq="TARGET_MOD="+target_mod+","+ "DURATION="+duration
    #match appliance
    #print "rcved"
    for app_name in cell.keys():
        l=app_name.split(".")
        app_type=l[1]
        app_user_group=l[2].split(" | ")[0]
        app_cls=l[2].split(" | ")[1]
        app_n=l[3]
        if app_type!=target_app:continue#appliance type must be matched
        if app_user_group==agent_type and app_cls=="personal" and app_n==agent_n:
            cell[app_name].rcv(rq)#use own
            print agent+" using personal "+app_type+" at time "+str(t)
            break
        elif app_user_group==agent_type and app_cls=="personal" and app_n!=agent_n:
            continue#use other's, not permitted
        elif app_user_group==agent_type and app_cls!="personal":
            cell[app_name].rcv(rq)#use group shared
            print agent+" using group shared "+app_type+" at time "+str(t)

```

```

        break
    elif app_user_group!=agent_type and app_cls=="personal":
        continue#use other group member's, not permitted
    elif app_user_group!=agent_type and app_cls!="personal":
        if app_user_group=="all":#use gloabal shared
            cell[app_name].rcv(rq)
            print agent+" using gloabal shared "+app_type+" at time "+str(t)
            break
        elif app_user_group!="all":#use other group shared,not this app
            continue
    #ignore req

#create objs for appliance in cells
path1=os.getcwd()+os.sep+"profiles_appliance"+os.sep
profiles=os.listdir(path1)
cells={}
for profile in profiles:
    l=profile.split(".")
    if cells.has_key(l[4]+"."+l[5])==False:
        cells[l[4]+"."+l[5]]=[]
        cells[l[4]+"."+l[5]].append(profile)
for k in cells.keys():
    cell={}
    for i in cells[k]:
        cell[i]=appliance.appliance(path1+i,"off")
    cells[k]=cell
#read req_log and integrate it into a sorted one
path2=os.getcwd()+os.sep+"logs_req"+os.sep
logs=os.listdir(path2)
cells_log={}
for log in logs:
    l=log.split(".")
    if cells_log.has_key(l[0]+"."+l[1])==False:
        cells_log[l[0]+"."+l[1]]=[]
        cells_log[l[0]+"."+l[1]].append(log)
for cell_n in cells_log.keys():
    cell_log=[]
    for cell_day in cells_log[cell_n]:
        lines=open(path2+cell_day,"r").readlines()
        for line in lines:
            day=int(cell_day.split(".")[3])-1
            new_line=re_time(line,day)[:1]
            cell_log.append(new_line)
        cells_log[cell_n]=sort_log(cell_log)
#create output logs
path3=os.getcwd()+os.sep+"data"+os.sep
outputs={}
for i in profiles:
    outputs[i]=[]
#####RUN#####

```

```

#get days
for i in open("init.cfg",'r').readlines():
    if i.split("=")[0]=="DAYS":
        days=int(i.split("=")[1])
#delay phases
for loc in cells.keys():
    for app in cells[loc].keys():
        for i in range(random.randint(0,86400)):
            cells[loc][app].run()
for loc in cells.keys():
    for app in cells[loc].keys():
        cells[loc][app].new_day()
#run for every second
for timer in range(1,days*86400+1):
    #timer increase
    #print "Time: "+str(timer)+"/"+str(days*86400)
    timer+=1
    for loc in cells.keys():
        #print "Generating power data for %s" % loc
        for app in cells[loc].keys():
            #print "Generating power data for %s" % app
            cells[loc][app].run()
            outputs[app].append(cells[loc][app].pwr)
#send req
for loc in cells_log.keys():
    if len(cells_log[loc])==0: continue
    if t(cells_log[loc][0])<=timer:
        req=cells_log[loc].pop(0)
        #REQ HANDLING HERE
        req_handler(cells[loc],req,timer)
#####WRITE DATA#####
print "=====
remove_files("data")
for output in outputs.keys():
    f=open(path3+output,"w")
    for i in range(len(outputs[output])):
        f.write(str(i)+","+str(outputs[output][i]))
        f.write("\n")
    f.close()
    print "Writing data file %s" %output
print "Data files generated"

```

Part C: the monitored appliances are:

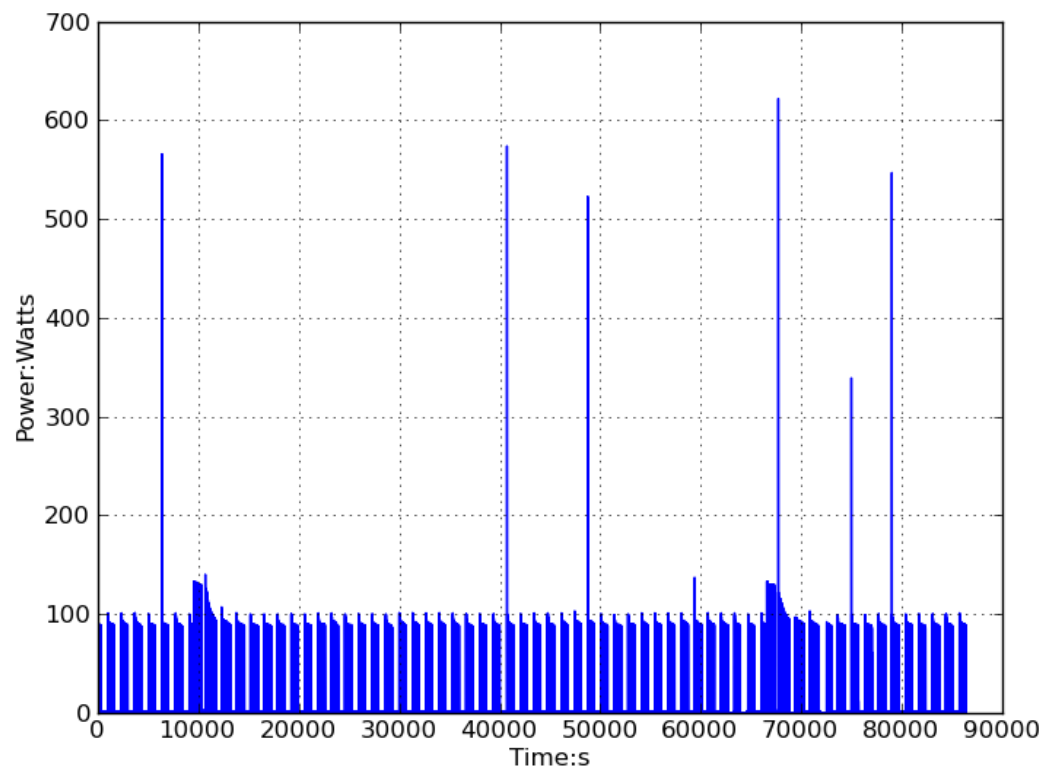


Fig.6.1. Power curve of refrigerator

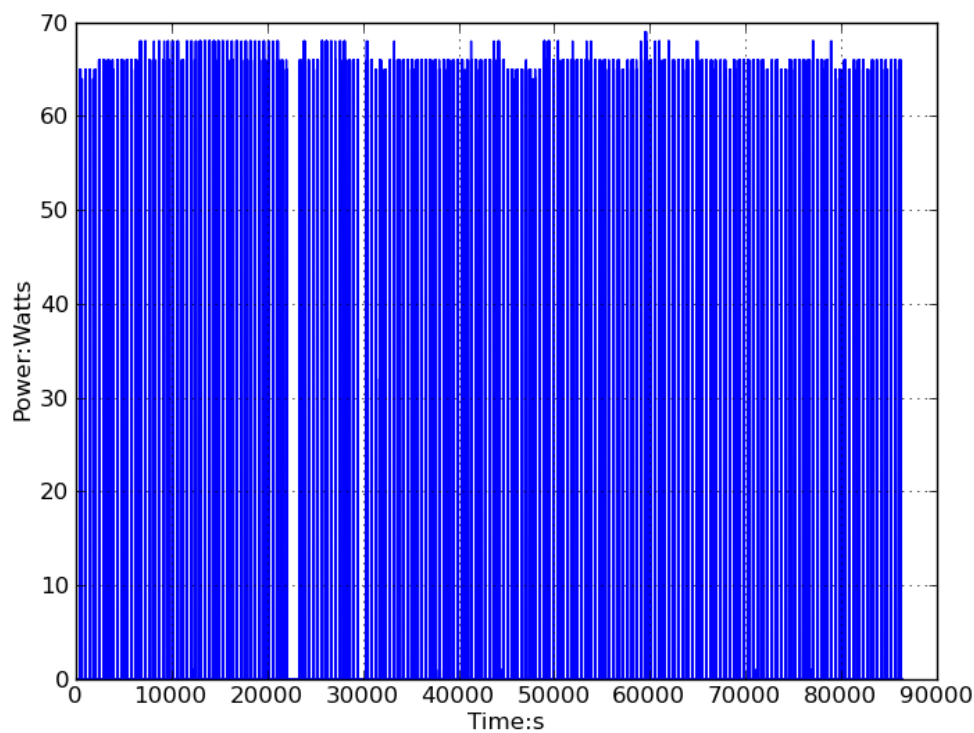


Fig.6.2. Power curve of kettle

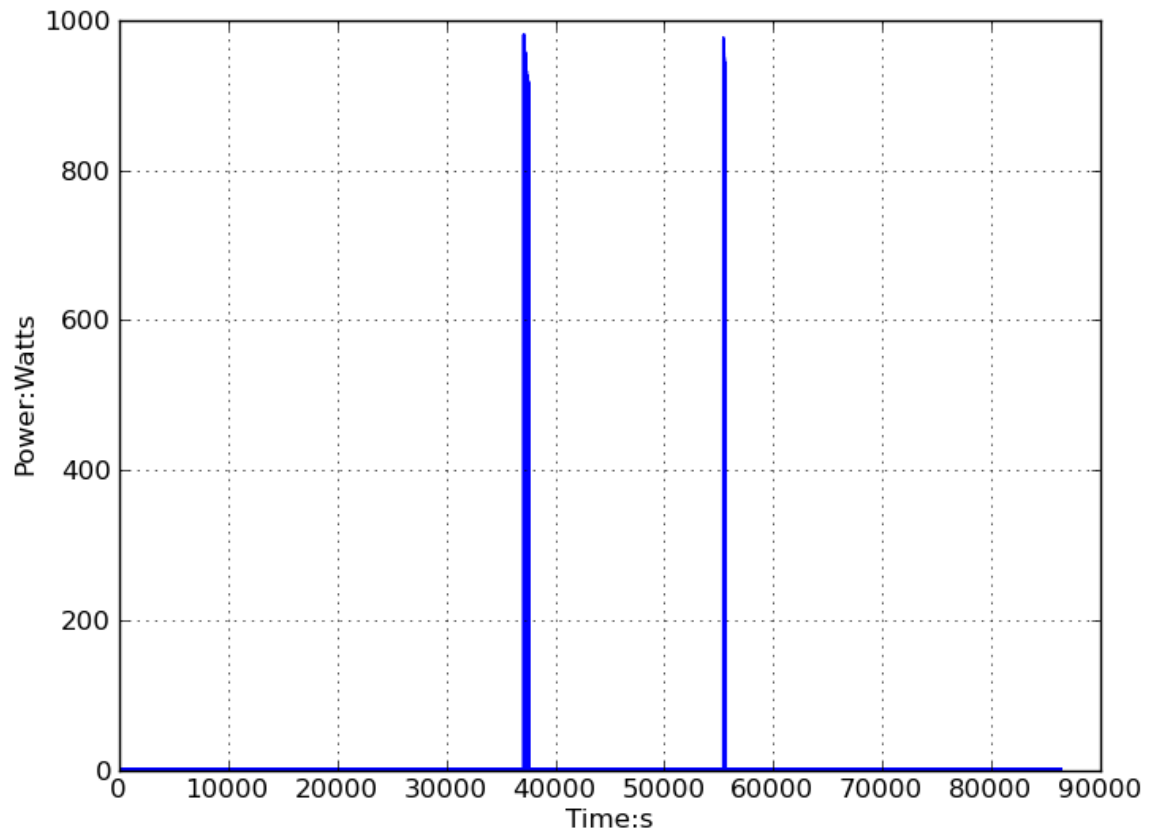


Fig.6.3. Power curve of microwave oven

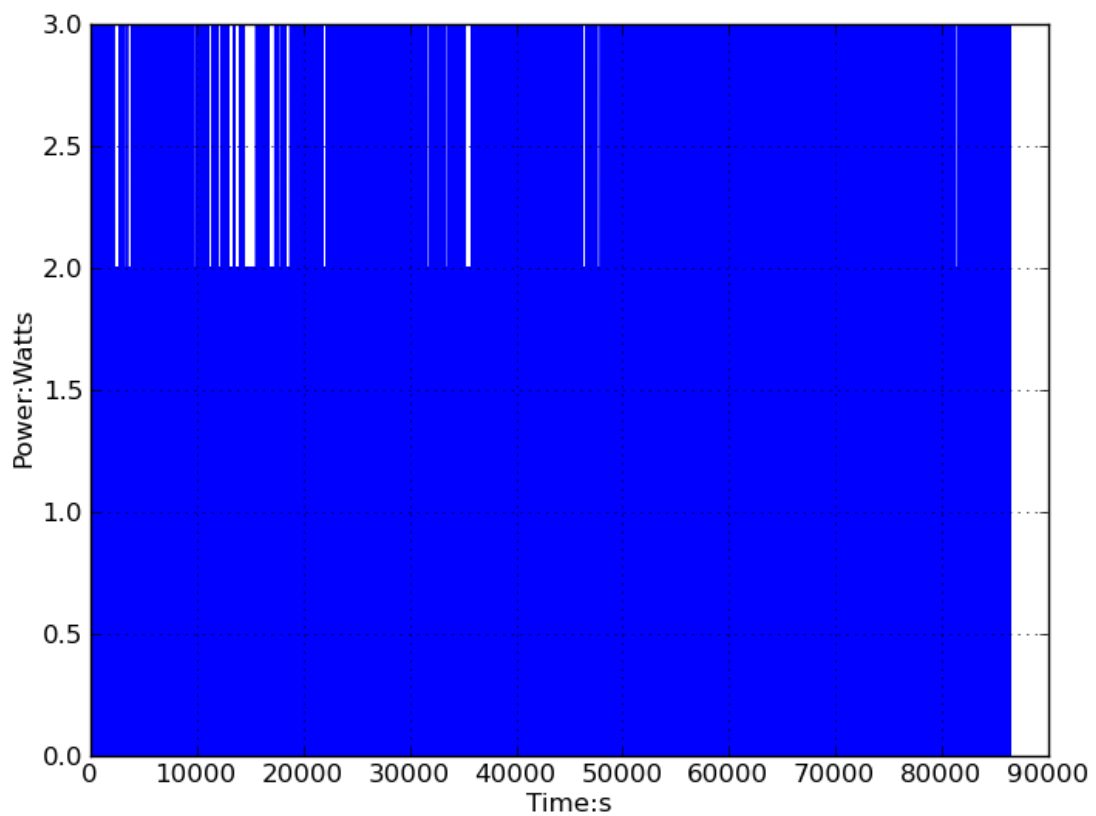


Fig.6.4. Power curve of router

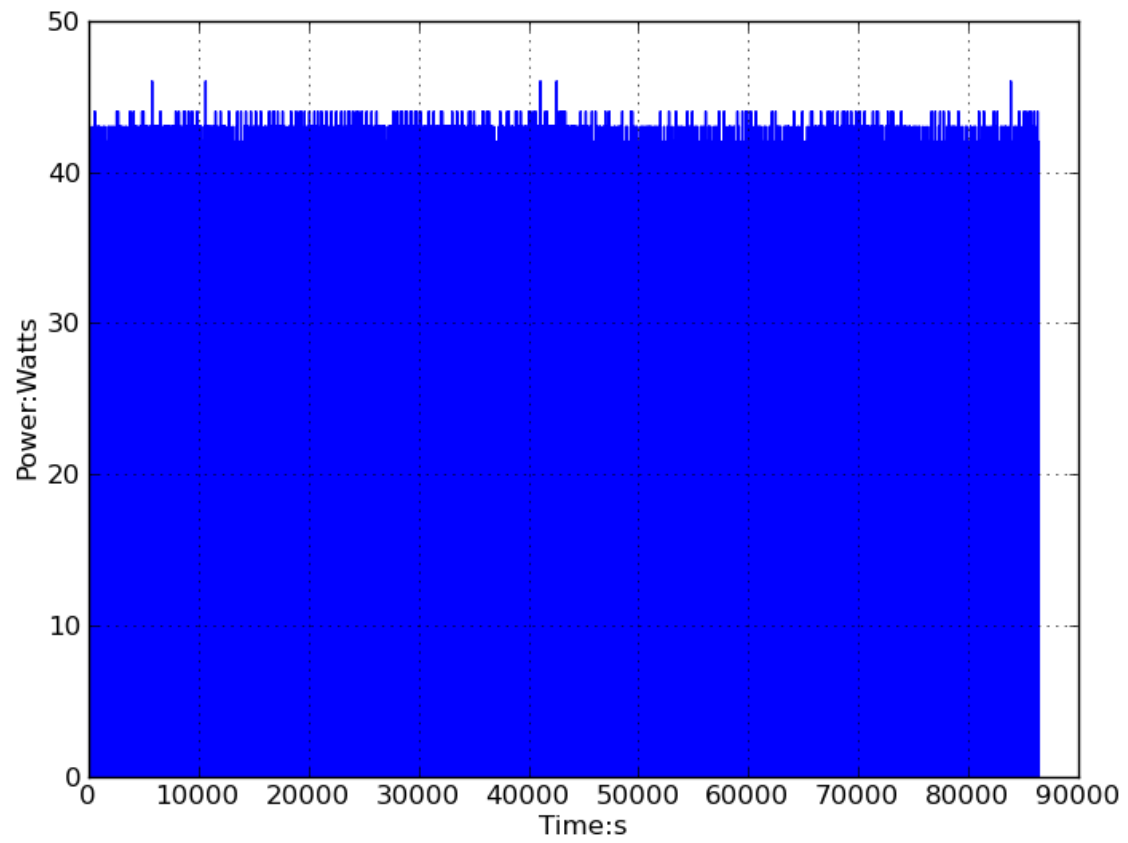


Fig.6.5. Power curve of server

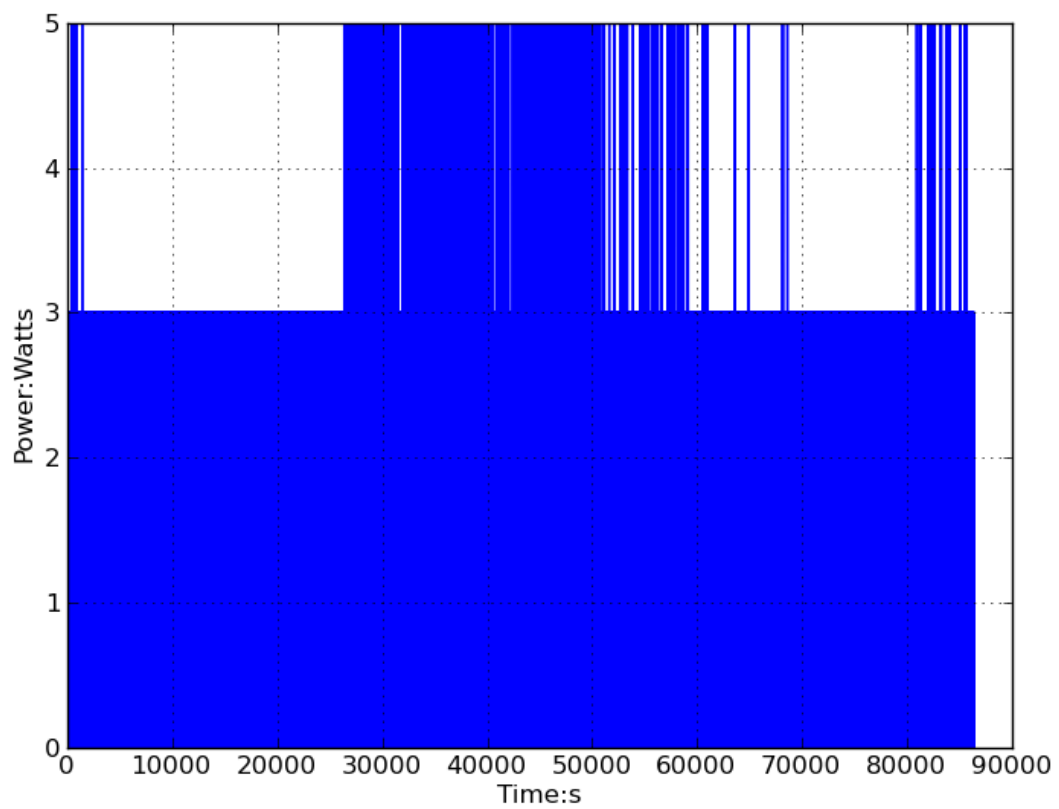


Fig.6.6. Power curve of phone

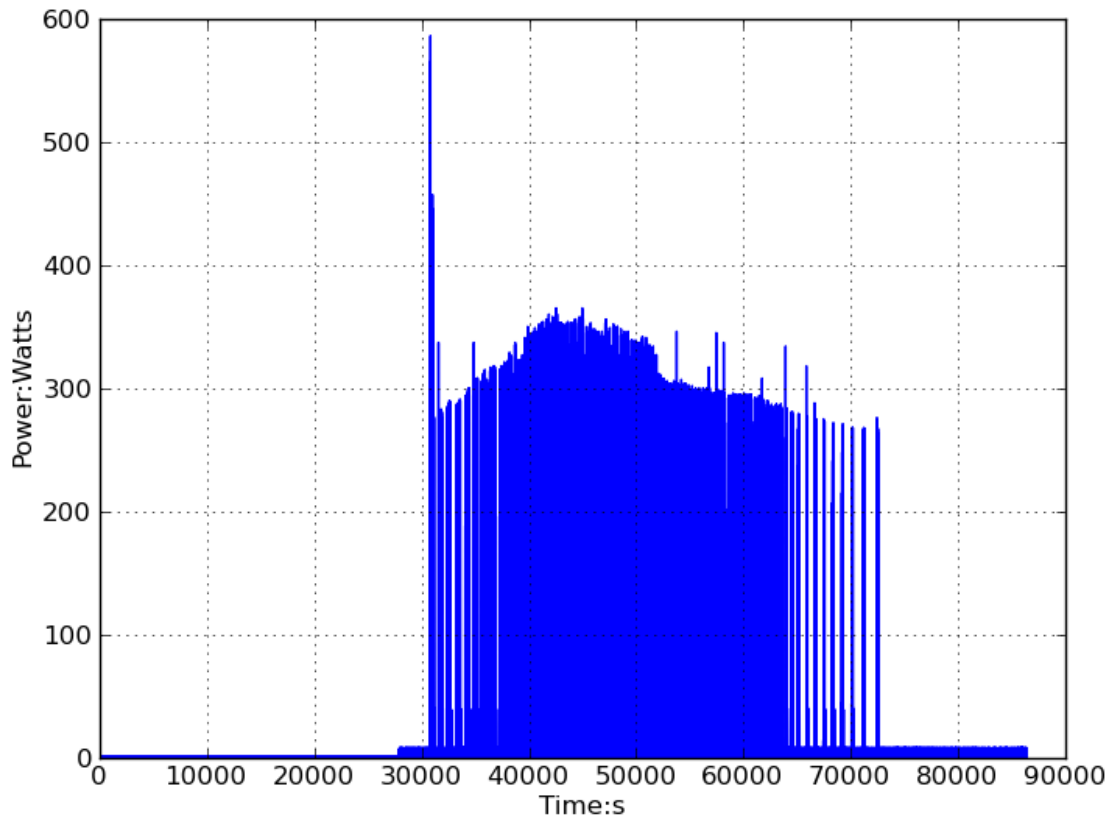


Fig.6.7. Power curve of air conditioner

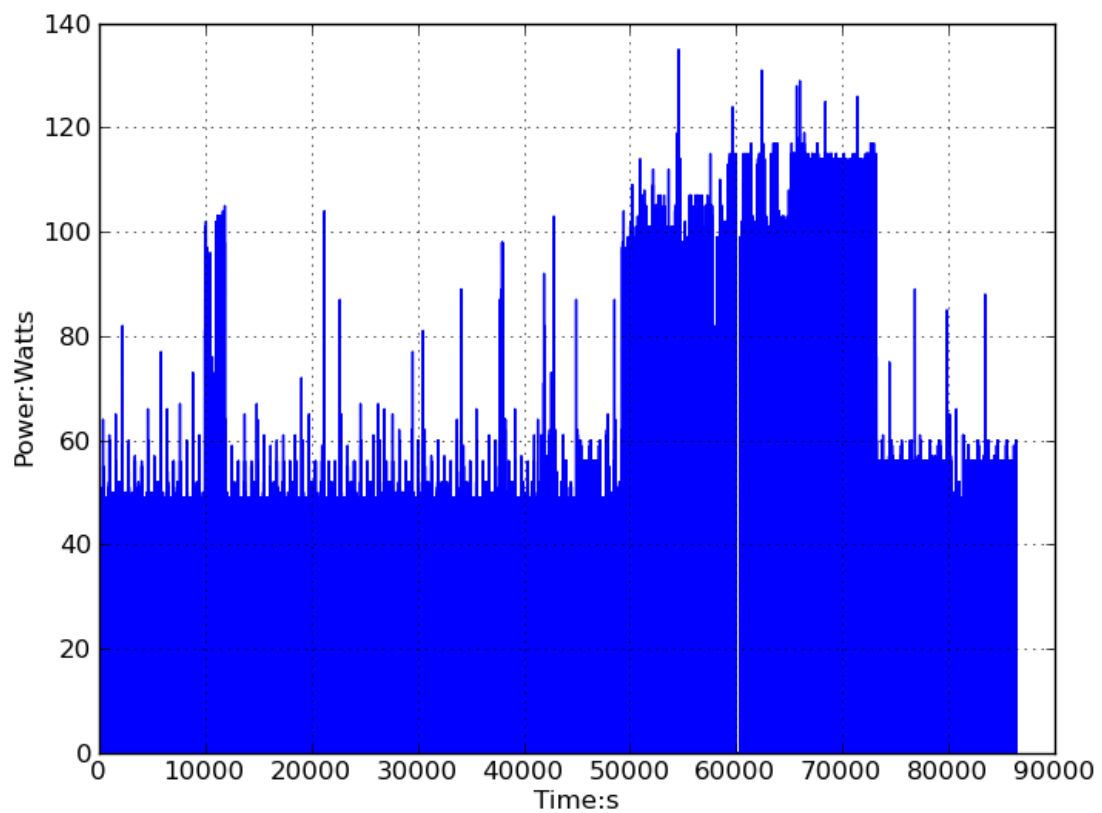


Fig.6.8. Power curve of desktop PC

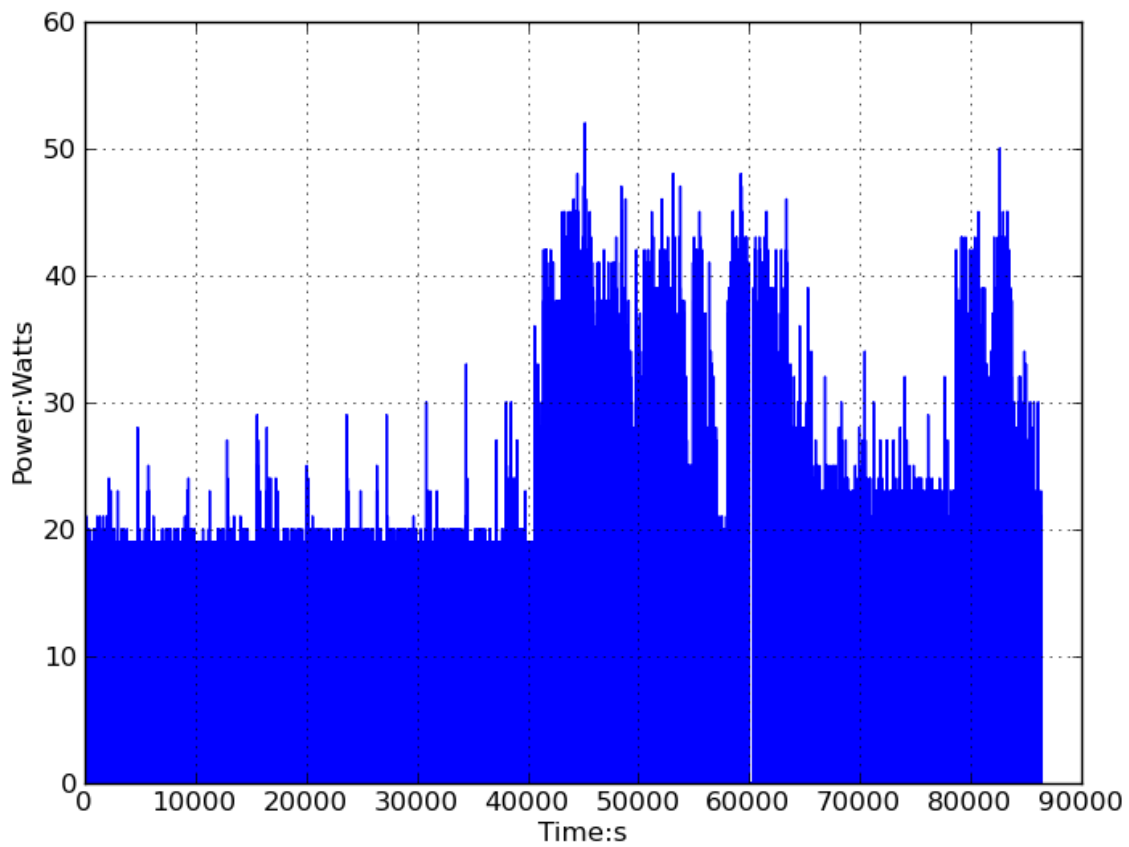


Fig.6.9. Power curve of laptop PC

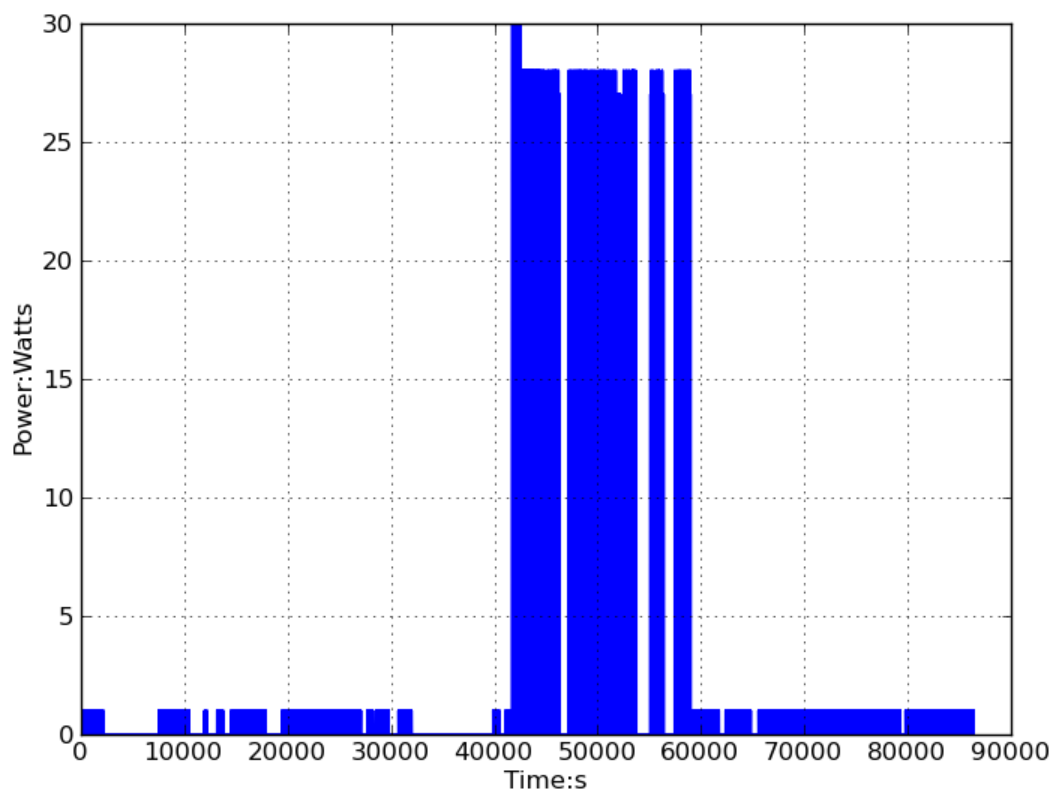


Fig.6.10. Power curve of display

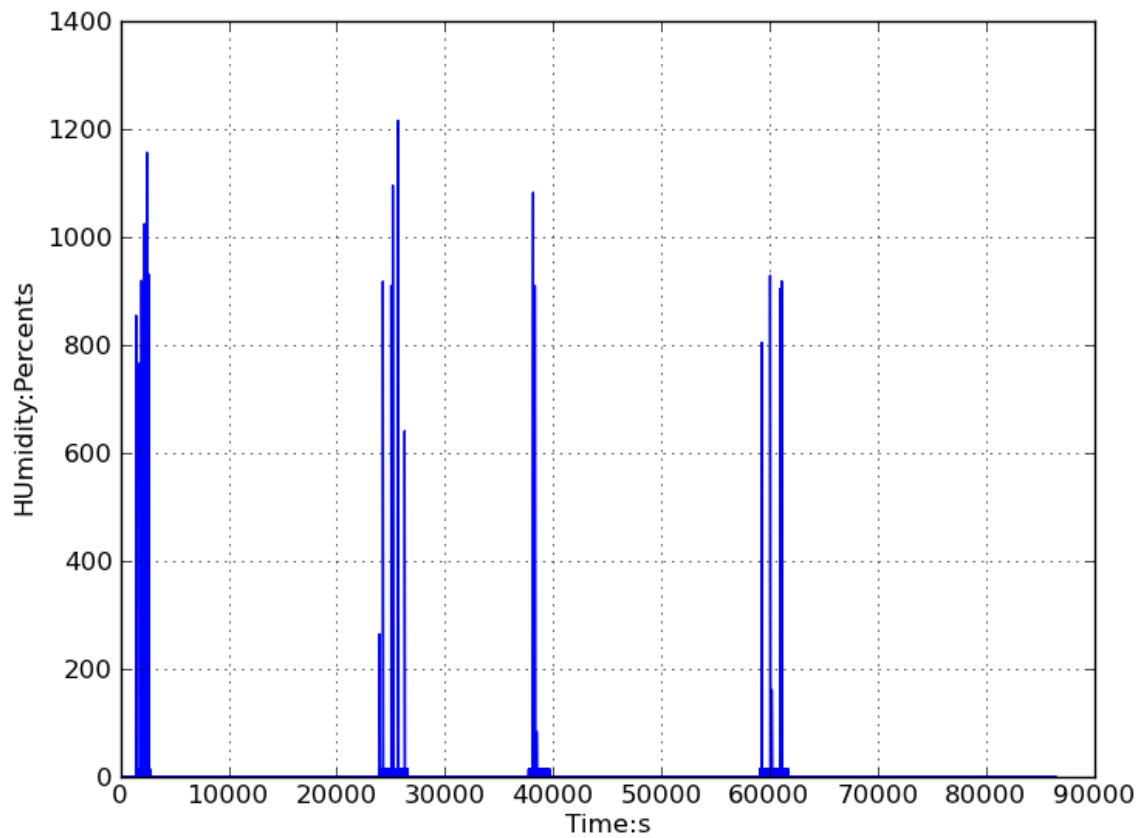


Fig.6.11. Power curve of printer

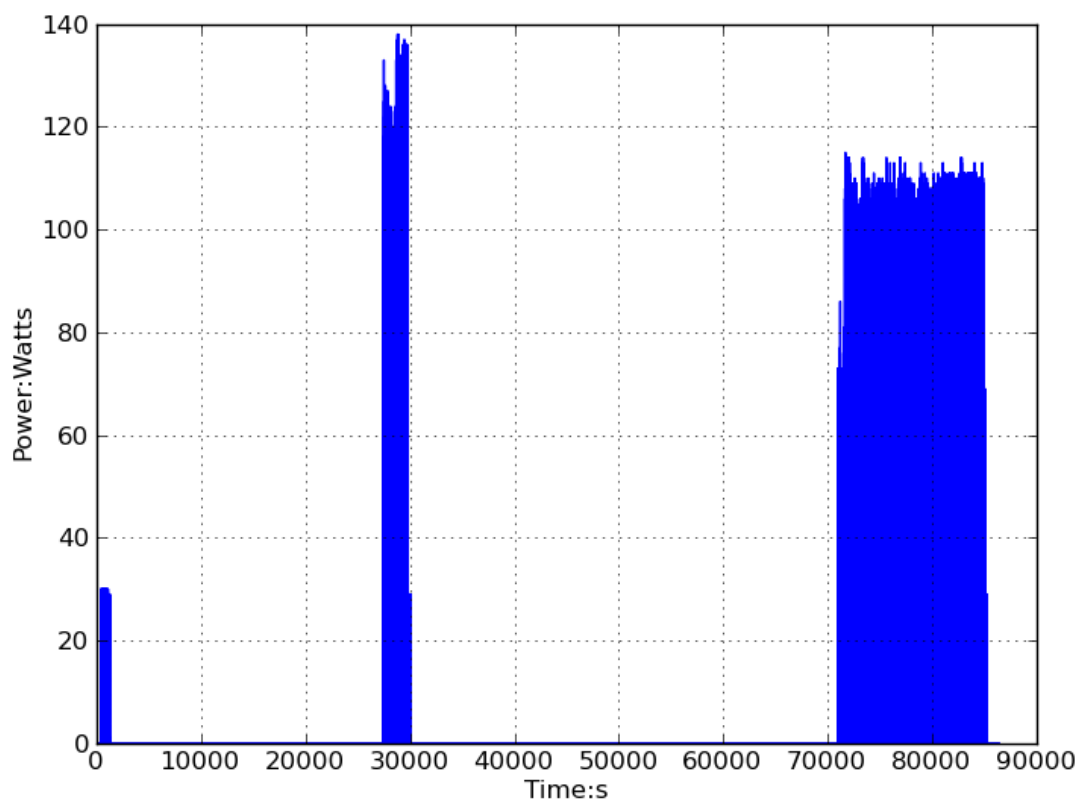


Fig.6.12. Power curve of TV

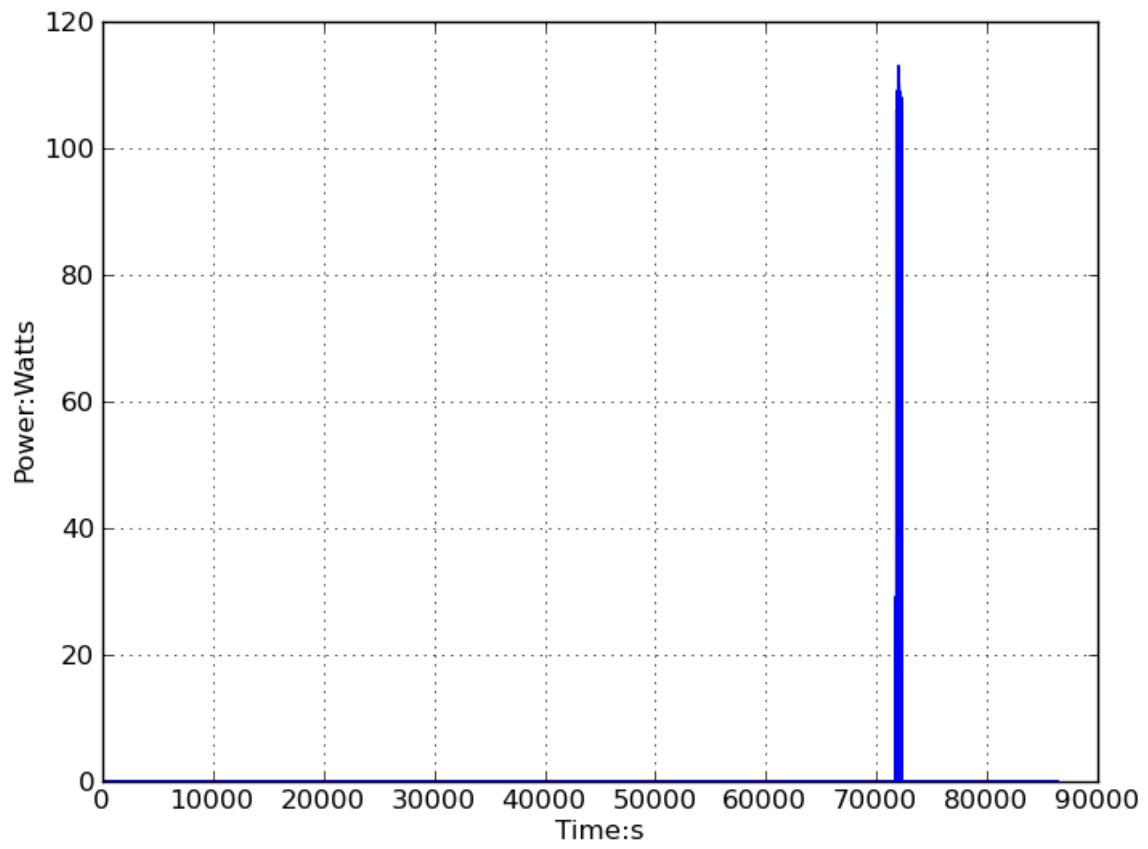


Fig.6.13. Power curve of TV recorder

Part D: template profiles for each appliance:

Refrigerator

node_table=off:0,0.1;stdby:480,0.3;w1:1200,0.3;w2:1200,0.5;w3:3600,0.3
transition_chart=off,stdby:0,0;stdby,w1:0,0;w1,stdby:0,0;stdby,w2:52000,0;w2,w3:0,0;w3,stdby:0,0

pcf_params=off:0 | 0/0/+ | 0,0,0;stdby:0 | 5/0/+ | 0,0,0;w1:1 | 95,10,-0.004/0.01/+ | 430,0.2,0.0002;w2:1 | 140,10,-0.008/0.01/+ | 430,0.2,0.0001;w3:1 | 100,55,-0.002/0.01/+ | 0,0,0

Kettle

node_table=off:360,0.3;on:130,0.3
transition_chart=off,on:0,0;on,off:0,0

pcf_params=off:0 | 0/0/+ | 0,0,0;on:0 | 67/0.03/+ | 0,0,0

Microwave oven

node_table=off:n/a;on:0,0
transition_chart=off,on:0,0;on,off:0,0

route_table=on,off,on:0,0

pcf_params=off:0 | 0/0/+ | 0,0,0;on:0 | 900/0.05/+ | 0,0,0

Router

node_table=off:0,0;w1:40000,0.5;w2:30000,0.5
transition_chart=off,w1:0,0;w1,w2:0,0;w2,w1:0,0

pcf_params=off:0 | 0/0/+ | 0,0,0;w1:0 | 3/0/+ | 0,0,0;w2:0 | 3/0/+ | 5,0,0.001

Server

node_table=off:400,0.1;on:40000,0.5
transition_chart=off,on:0,0;on,off:0,0

pcf_params=off:0 | 0/0/+ | 0,0,0;on:0 | 43/0.07/+ | 0,0,0

Phone

node_table=off:0,0;stdby:25000,0.3;charging:50000,0.3
transition_chart=off,stdby:0,0;stdby,charging:0,0;charging,stdby:0,0

pcf_params=off:0 | 0/0/+ | 0,0,0;stdby:0 | 3/0/+ | 0,0,0;charging:0 | 3/0/+ | 5,0,0.001

Air conditioner

node_table=off:0,0;w1:12000,0.2;w2:0,0;working:0,0
transition_chart=off,w1:n/a;w1,off:0,0;w1,w2:0,0;w2,working:0,0;working,w1:0,0

route_table=working,off,w1:0,0;working,w1,w2:2500,0.1;working,w2,working:1000,0

pcf_params=off:0 | 0/0/+ | 0,0,0;w1:0 | 10/0.2,0.01/+ | 0,0,0;w2:0 | 500/0.2,0.03/+ | 0,0,0;working:2 | 260,90,46000,10000/0.05,0.01/+ | 0,0,0

Desktop PC

node_table=off:n/a;stdby:108000,0;w1:0,0;w2:0,0
transition_chart=off,stdby:n/a;stdby,off:0,0;stdby,w1:n/a;stdby,w2:n/a;w1,stdby:0,0;w1,w2:n/a;w2,stdby:0,0;w2,w1:n/a

route_table=w1,off,stdby:0,0;w1,stdby,w1:0,0;w1,w2,w1:0,0;w2,off,stdby:0,0;w2,stdby,w2:0,0;w2,w1,w2:0,0

pcf_params=off:0 | 0/0/+ | 0,0,0;stdby:0 | 50/0/+ | 50,0.6,0.005;w1:0 | 100/0/? | 100,0.05,0.01;w2:0 | 110/0/? | 115,0.1,0.01

Laptop PC

node_table=off:n/a;stdby:45000,0;w1:0,0;w2:0,0
transition_chart=off,stdby:n/a;stdby,off:0,0;stdby,w1:n/a;stdby,w2:n/a;w1,stdby:0,0;w1,w2:n/a;w2,stdby:0,0;w2,w1:n/a

route_table=w1,off,stdby:0,0;w1,stdby,w1:0,0;w1,w2,w1:0,0;w2,off,stdby:0,0;w2,stdby,w2:0,0;w2,w1,w2:0,0

pcf_params=off:0 | 0/0/+ | 0,0,0;stdby:0 | 2/0/+ | 0,0,0;w1:0 | 20/0/+ | 20,0.5,0.01;w2:0 | 30/0/+ | 35,0.14,0.01

Display

node_table=off:0,0;stdby:n/a;working:0,0
transition_chart=off,stdby:0,0;stdby,working:0,0;working,stdby:0,0

route_table=working,stdby,working:0,0;working,off,stdby:0,0

pcf_params=off:0 | 0/0/+ | 0,0,0;stdby:0 | 1/0/+ | 0,0,0;working:0 | 27/0/+ | 29,0.03,0.005

Printer

node_table=off:n/a;ready:1000,0.5;working:0,0
transition_chart=off,ready:0,0;ready,off:0,0;ready,working:n/a;working,ready:0,0

route_table=working,off,ready:0,0;working,ready,working:100,0.5

pcf_params=off:0 | 0/0/+ | 0,0,0;ready:0 | 20/0/+ | 0,0,0;working:0 | 900/0.3/+ | 0,0,0

TV

node_table=off:n/a:on:0,0

transition_chart=off,on:0,0;on,off:0,0

route_table=on,off,on:0,0

pcf_params=off:0 | 0/0,1/+ | 0,0,0;on:2 | 110,30,43200,10000/0.05,0.01/+ | 0,0,0

TV recorder

node_table=off:n/a:on:0,0

transition_chart=off,on:0,0;on,off:0,0

route_table=on,off,on:0,0

pcf_params=off:0 | 0/0/+ | 0,0,0;on:0 | 110/0.05/+ | 0,0,0