# Project Phase 1: **Internet of Things - Smarter Home Edition**

## Multi tiered Architecture:

The database is implemented in the back end tier and all the sensors and gateway are assumed to be in front end tier. All the sensors and devices either push events to gateway or pull events from sensors and devices. Gate way is made to communicate with the database to populate the database with all the push and pull events from sensors and devices at a particular time stamp. Gateway can query the database and get the current status of a device or sensor and can report the particular machine. Database is Synchronized to avoid multiple threads accessing at same time.

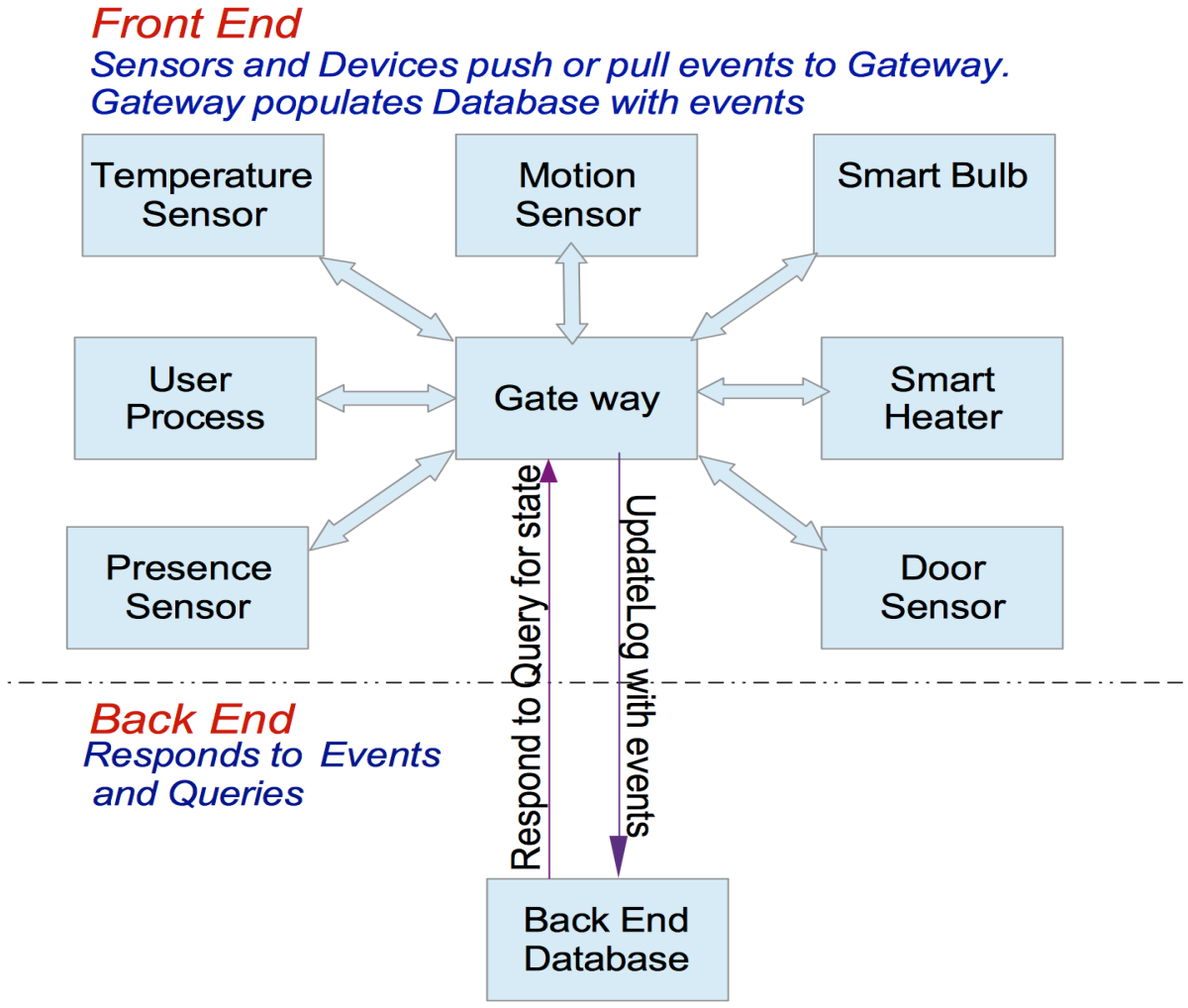



**Figure** Error! No sequence specified. **Multi tiered Architecture**

**Leader Selection Algorithm – Ring Algorithm:**

Leader in this distributed system has to be selected to maintain time synchronization. Ring algorithm is used to select the leader. Each node in the system has information about its neighbors and can communicate with them. We can select any machine and initiate the ring selection algorithm. ***The initiator machine can start a token with its id included and pass it to the next machine. The next machine appends its id to the token and pass to its neighbor.*** When the token passes through all the machines and come back to initiator of election then the initiator can iterate through the token to find the machine with highest id and broadcast the result to other machines.

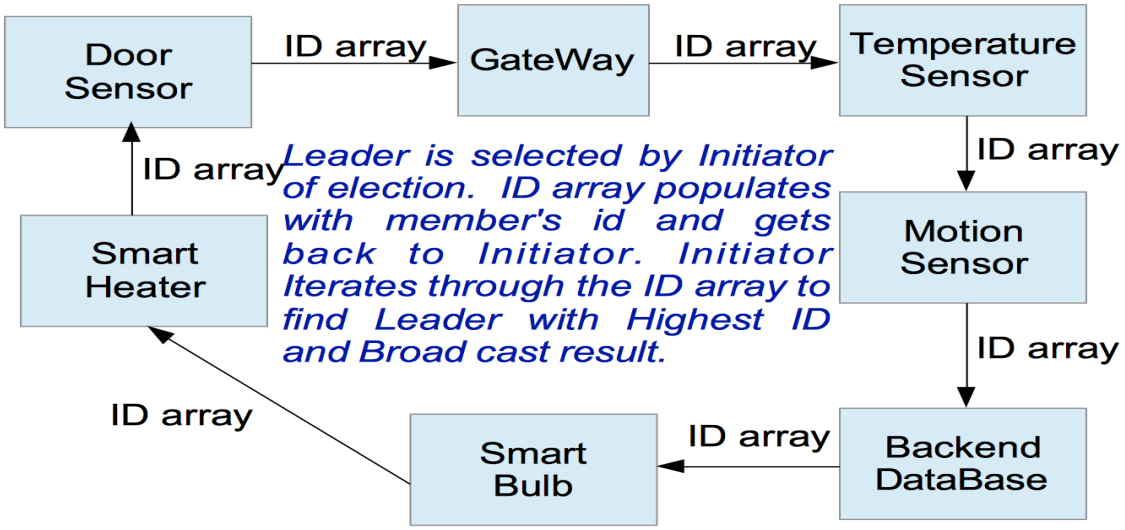



**Figure** Error! No sequence specified. **Ring Algorithm implementation**

**Handling Machine failure Ring Algorithm:**

If a Machine fails then the machine before the node failure performs the leader election and broadcast the result.

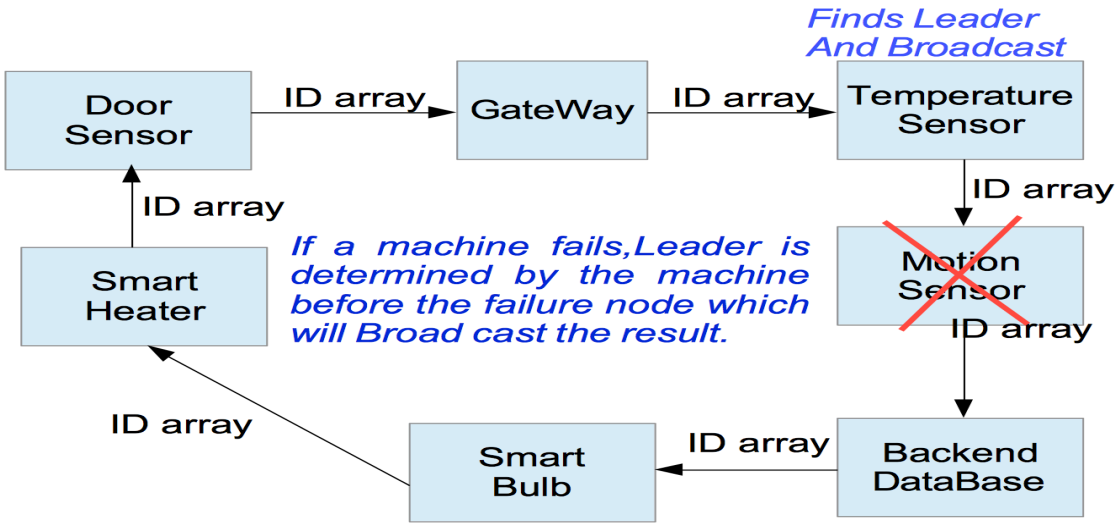



**Figure** Error! No sequence specified. **Handling machine failure in Ring Algorithm**

**Time Server and clock synchronization (Berkley Algorithm):**

The leader selected from the above-mentioned ring algorithm performs the clock synchronization. Once the leader is selected then the leader will act as a timeserver and requests the current time from the remaining machines. Leader will then compute the average of the time received from other machines and broadcast the average time to all the machines. All machines then compute the offset variable based on the difference between their current time and the time received from the master.

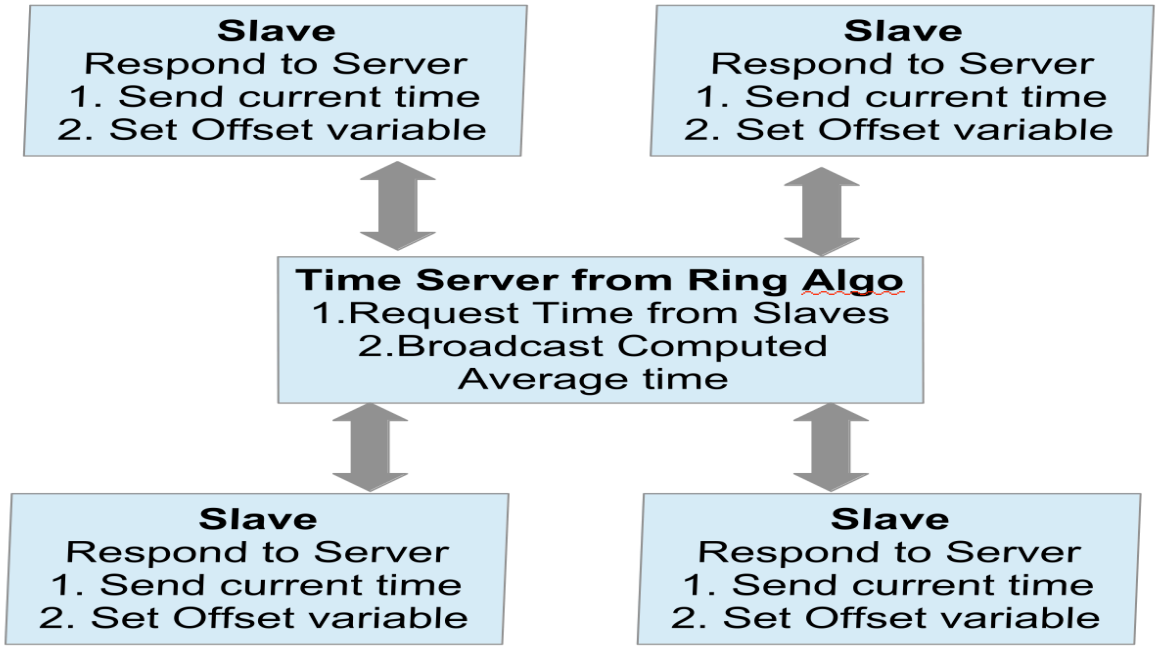


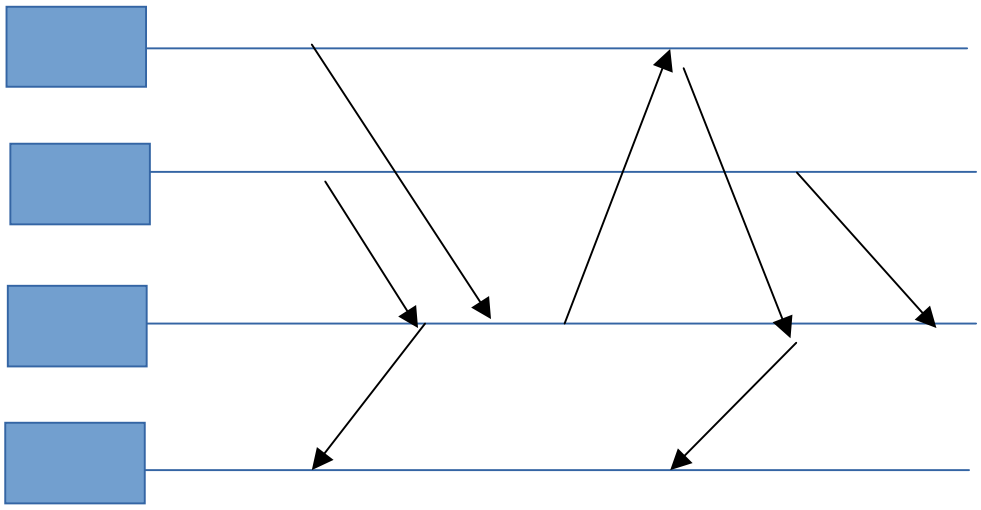

**Figure** Error! No sequence specified. **Time Server Implementation**

## Lamport's logic clock/vector clock algorithm

Clock syncronization can also be accomplished by logic clock, lamport clock which is useful for event order. For some concurrent events, we don't care about the order.
We only care about the event that communicates with each other.
If event a happened before event b, then logic value LC(a) < LC(b).
we use lamport logic clock to infer the order of all sensors and devices's events with gateways.

This basic lamport clock is implemented as Fig.5 which only shows four processes.
door, motion, gateway and bulb's logic clock values changes . Others processes will have similar rules.
1) initialize all the gateway, sensors and devices's LC as 0
2) then according to lamport's algorithm,
Whenever an event occurs locally at i, LC i = LC i +1
When i sends message to j, piggyback Lci
When j receives message from i
If LC j < LC i then LC j =max(LC i , LC j)+1

through this rule, we can get a series of event's value in gateway, for example
what is the LC value for motion is sensed, what is LC for door open sensed etc.
we record every event's clock value at gateway side termnial and log them including logictimestamp, deviceId, devicetype for sending message, event, and device's current status into the database file
the log format for logic clock in a different database file has the following format which is a little different from the dabase format from physical clock synchronization



## Event ordering

About event order, we can use physical clock synchronization implemented in part1, as well as this, we can also use lamport logic clock designe in part2.

Takes motion sensor and door sensor data to infer when someone entered or left the house. For instance, if motion is sensed first and the door open event happens later, it follows that someone has exited the home. Conversely, if the door open event is seen first and motion is detected later, someone has entered the house.
We used this to design a testcase called lab2-test-input.csv to automatically inferred user came home and left home

Also, we use presence sensor to automatically randomly push event to gateway, if gateway received presence "on", we inferred its user's activity.
If gateway receive presence "off", we inferred its intruder's activity.
The test case would be describer below.

**Implementation Platform details**

→ Java
→ RMI
→ Linux Platform supported scripts.
**Instructions for running the Code and Test scripts:**

***The code can be can using three different scripts:***

1. **run-part1.sh** ----- To Manually enter data and test if Leader election(Ring Algorithm), Time server and Database are working. (Part 1 in Assignment)
2. **run-part2.sh** ----- To Manually enter data and test if Logical clock implementation and Database are working. (Part 2 in Assignment)
3. **run-test-case.sh** ----- To run test cases provided in a "**lab2-test-input.csv**" to simulate event ordering and the above parts automatically. This includes automatically simulating part1 and part2 manually tested above.

We have decentralized our code into various Java Packages where each Package corresponds either to a component (the gateway, a sensor or a device or Back end database). In order to avoid the complexity and exceptions due to the supporting code being in different packages, **In addition to the source code we have also provided the executable jar files for each component above described. We have Submitted Source code to verify code and Jar files to execute.**

**IP Address Recognition and Allocation:**
We are needed to provide only the IP Address of the Gateway in the Configuration file (**configips.csv**). The Default IP Address in configips.csv is local host. This IP Address is needed for all the other components. Each Component can figure out the value of its IP

Address when initiated and will register at the Gateway. Gateway stores the IP Address and can access the other components when required.

**Test-input file:**
The Test-input file "**lab2-test-input.csv**" can be used to assign test cases for the application when operated by running the automated scripts instead of user input of test cases.

**Jar files & command line arguments in various cases:**

***For Test cases file as Input(lab2-test-input.csv):*** **run-test-case.sh**
We have implemented the test-case provided and it will generate event log file **"event-log1.csv"** and "**event-log2.csv**" in the same Directory of Execution from database. The following Jar files will take the command line arguments as Path to the Configuration file and Path to Test-input file ***(lab2-test-input.csv)*** as arguments. **Please place all the Jar files, configips.csv and lab2-test-input.csv in the same Directory.**

```
gnome-terminal -x sh -c "java -jar GatewayServer.jar configips.csv lab2-test-input.csv;
bash"&
sleep 2
gnome-terminal -x sh -c "java -jar tempeSensor.jar configips.csv lab2-test-input.csv; bash"&
sleep 1
gnome-terminal -x sh -c "java -jar HeaterSmart.jar configips.csv lab2-test-input.csv; bash"&
sleep 1
gnome-terminal -x sh -c "java -jar bulbSmart.jar configips.csv lab2-test-input.csv; bash"&
sleep 1
gnome-terminal -x sh -c "java -jar DoorSensor.jar configips.csv lab2-test-input.csv; bash"&
sleep 1
gnome-terminal -x sh -c "java -jar BackendDatabase.jar configips.csv lab2-test-input.csv;
bash"&
sleep 1
gnome-terminal -x sh -c "java -jar motionSensor.jar configips.csv lab2-test-input.csv; bash"&
sleep 1
gnome-terminal -x sh -c "java -jar PresenceSensor.jar configips.csv lab2-test-input.csv;
bash"&
sleep 1
gnome-terminal -x sh -c "java -jar UserOperation.jar configips.csv; bash"&
```

***For manual Test cases entered as Input by user:***
***Part1--*** **run-part1.sh** ***(Ring Algorithm & Time server):***
The user can manually enter the various events in various terminals (if using a singe machine in the same machine or using different in different machines). In this case, all the following Jar files will take the command line argument as Path to the Configuration file(**configips.csv**) and ***"part1"*** to differentiate between part1 and part2 implementation . This can run in different Machines by users manually.

```
gnome-terminal -x sh -c "java -jar GatewayServer.jar configips.csv part1; bash"&
sleep 3
gnome-terminal -x sh -c "java -jar tempeSensor.jar configips.csv part1; bash"&
gnome-terminal -x sh -c "java -jar HeaterSmart.jar configips.csv part1; bash"&
gnome-terminal -x sh -c "java -jar bulbSmart.jar configips.csv part1; bash"&
gnome-terminal -x sh -c "java -jar DoorSensor.jar configips.csv part1; bash"&
```

```
gnome-terminal -x sh -c "java -jar BackendDatabase.jar configips.csv part1; bash"&
gnome-terminal -x sh -c "java -jar motionSensor.jar configips.csv part1; bash"&
gnome-terminal -x sh -c "java -jar PresenceSensor.jar configips.csv part1; bash"&
gnome-terminal -x sh -c "java -jar UserOperation.jar configips.csv part1; bash"&
```

Execution of these commands will open 9 Terminals in which user can enter the test cases as illustrated using Sequence Diagrams in this document. An output file **"event-log1.csv"and/or "event-log2.csv" database file** is created by the Gateway with the tuples containing time stamp values, deviceId, deviceType, currentState, event and inferred Activity from sensors and devices. **You can initiate leader election in any one the machines or processes.**

***Part2--* run-part2.sh *(Logical clock):***

The user can manually enter the various events in various terminals (if using a singe machine in the same machine or using different in different machines). In this case, all the following Jar files will take the command line argument as Path to the Configuration file(**configips.csv**) and "***part2"*** to differentiate between part1 and part2 implementation . This can run in different Machines by users manually.

```
gnome-terminal -x sh -c "java -jar GatewayServer.jar configips.csv part2; bash"&
sleep 1
gnome-terminal -x sh -c "java -jar tempeSensor.jar configips.csv part2; bash"&
sleep 1
gnome-terminal -x sh -c "java -jar BackendDatabase.jar configips.csv part2; bash"&
sleep 1
gnome-terminal -x sh -c "java -jar HeaterSmart.jar configips.csv part2; bash"&
sleep 1
gnome-terminal -x sh -c "java -jar bulbSmart.jar configips.csv part2; bash"&
sleep 1
gnome-terminal -x sh -c "java -jar DoorSensor.jar configips.csv part2; bash"&
sleep 1
gnome-terminal -x sh -c "java -jar motionSensor.jar configips.csv part2; bash"&
sleep 1
gnome-terminal -x sh -c "java -jar PresenceSensor.jar configips.csv part2; bash"&
sleep 1
gnome-terminal -x sh -c "java -jar UserOperation.jar configips.csv part2; bash"&
```

Execution of these commands will open 9 Terminals in which user can enter the test cases as illustrated using Sequence Diagrams in this document. An output file **"event-log1.csv"and/or "event-log2.csv" database file** is created by the Gateway with the tuples containing time stamp values, deviceId, deviceType, currentState, event and inferred Activity from sensors and devices.

**Executable Script files:**

***run-part1.sh & run-part2.sh*** : This Script files corresponds to the automation of the "***For manual Test cases entered as Input by user"*** case discussed above. This script is executable and has all permissions. User Just need to run this script and Terminals Pop up and the user need to enter various values depending on the statements in the terminal and the Application can be tested. **User Input Needed. The first or second line output in terminal will convey the operation performed by that process. See Appendix: 1 and Appendix 2**

***lab2-run-test-case.sh:*** This Script file corresponds to the automation of the "***For Test cases***

***file as Input"*** case discussed above. This script is executable and has all permissions. The script will execute in different Terminals and an output file **"event-log1.csv"and/or "event-log2.csv" database file** is created by the Gateway with the tuples containing time stamp values, deviceId, deviceType, currentState, event and inferredActivity from sensors and devices. **No Need of User Input. Please wait till all the execution completes in 3 terminals and the output from the programs stop (Doesn't move). See Appendix: 2**

**Test Cases Sequence Diagrams:**

Leader election process is discussed in the above description (Leader Selection Algorithm – Ring Algorithm)

1. **Events being logged into Backend database:**

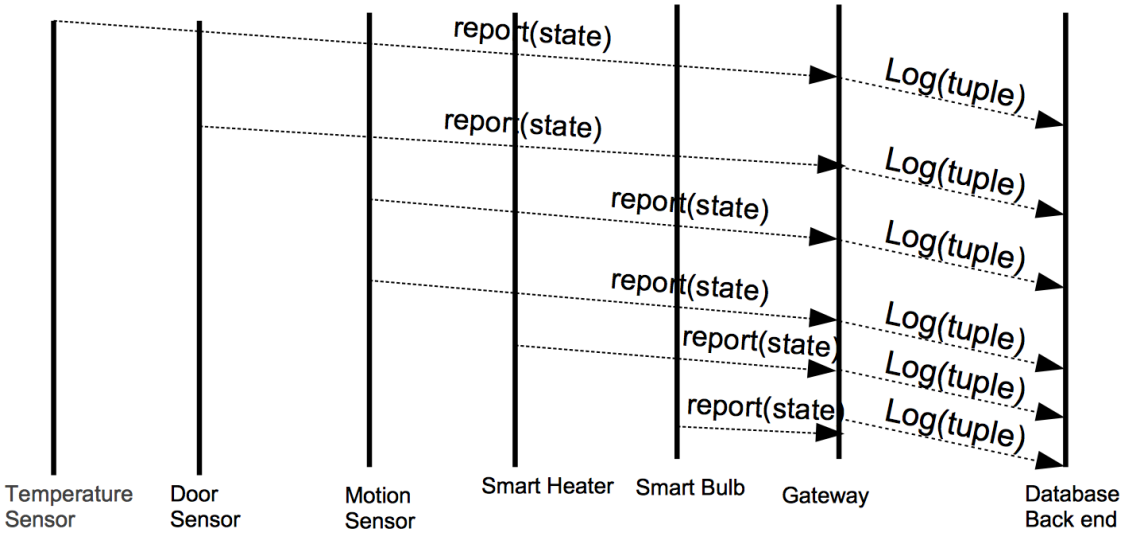



**Figure 1 Logging Events into Database**

2. **Broadcasting election result and Time server functionality:**

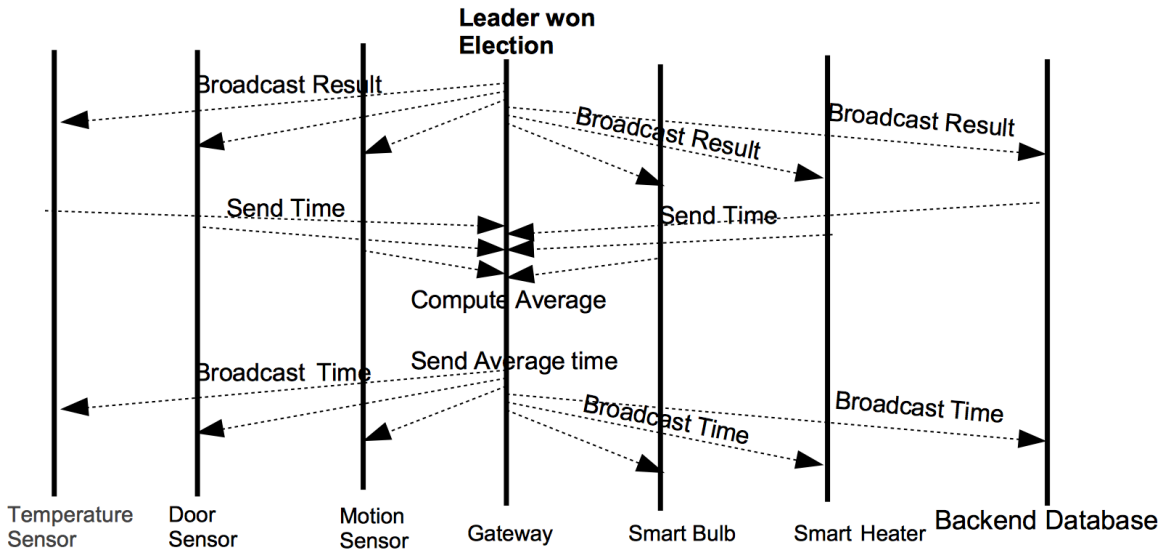



**Figure 2** Broadcast Result and Berkley Algorithm

## 3. Event ordering:

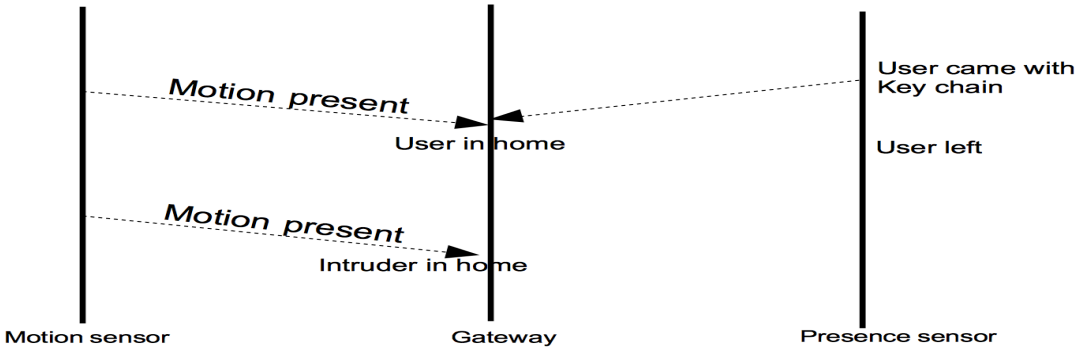



**Figure 3** Event Ordering

## 4. Query Database:

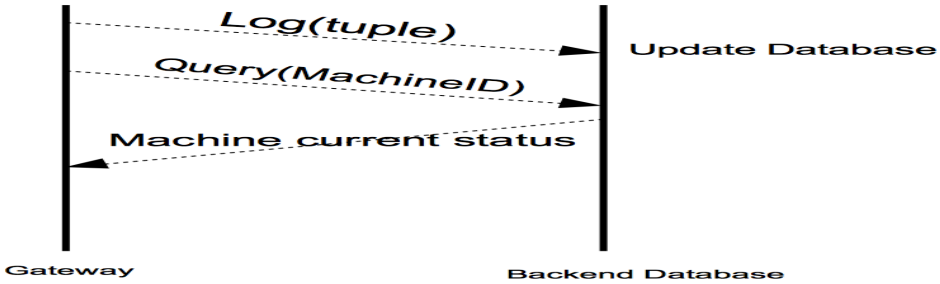



**Figure 4** Query Database

## Performance Analysis:

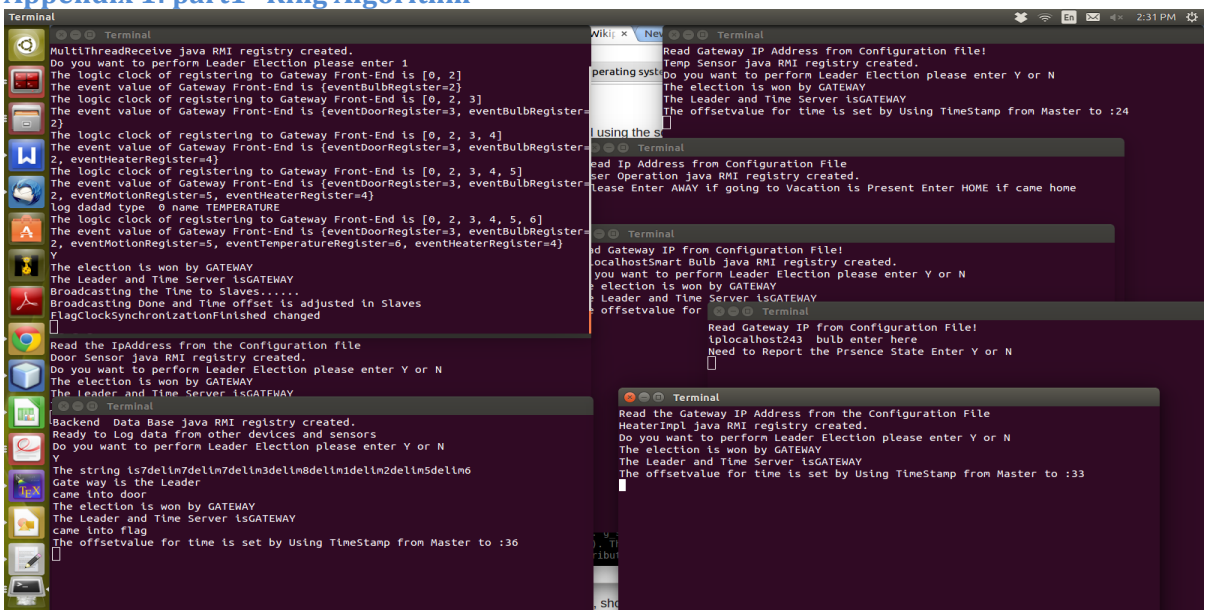
We have performed experiments on the performance by measuring the delay from between Gateway and the Sensors.
→Push Performance delay test which Gateway Received push temperature from Motion sensor.
→Pull Performance delay test which Gateway pull temperature from temperature sensor

| Interval | 500ms(average) | 1000ms | 2000ms | 5000ms | 10000ms |
|---|---|---|---|---|---|
| push(report) delay(milliseconds) | 3.454 | 2.898 | 3.572 | 3.452 | 3.12.3 |
| Pull dealy (query(milliseconds) | 4.224 | 3.554 | 4.232 | 4.522 | 4.134 |
| Broad cast delay (milliseconds) | 3.756 | 2.967 | 3.421 | 3.653 | 2.983 |

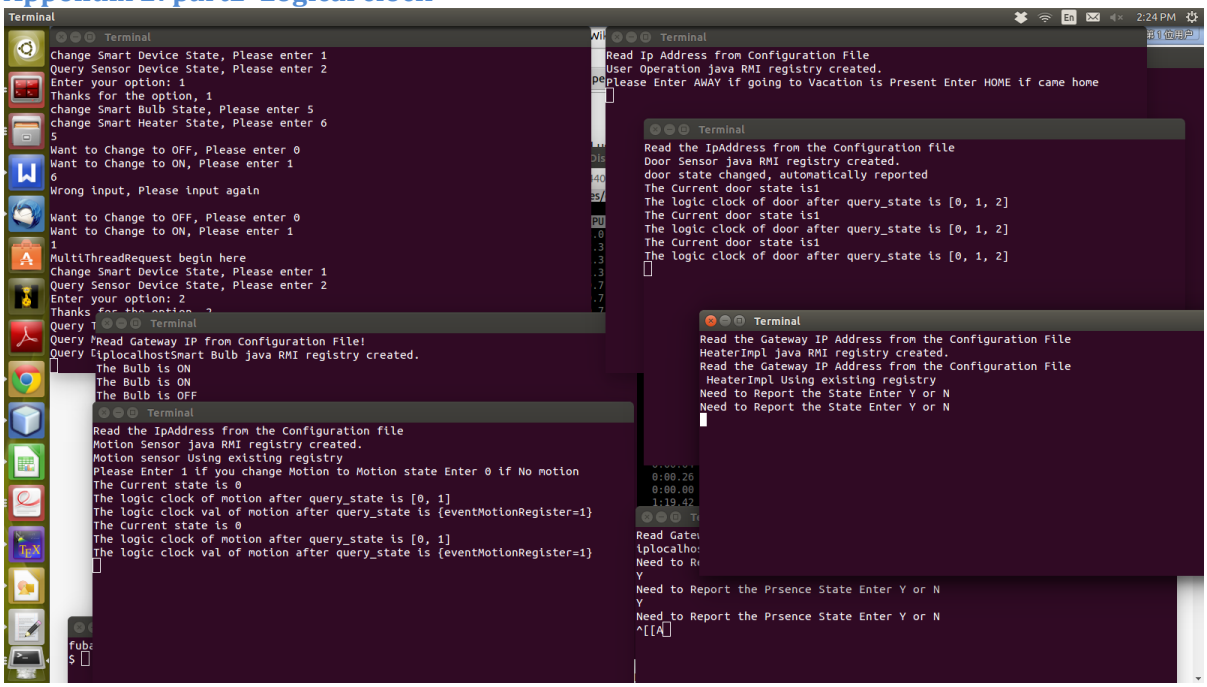
From basic statistics of this table, the delay of push and pull are very small where as pull delay is a more than push.

**Conclusion:**
→ Application can be used to run in Automated test case Mode test cases provided as a file.
→ Application can also run in Manual mode where user has to interact from the terminal.
→ Generated an output file containing the log file tuples of time stamps and sensor values for automated mode.
→ we have extensively tested all the Test cases as mentioned above to ensure the application is executing as expected.

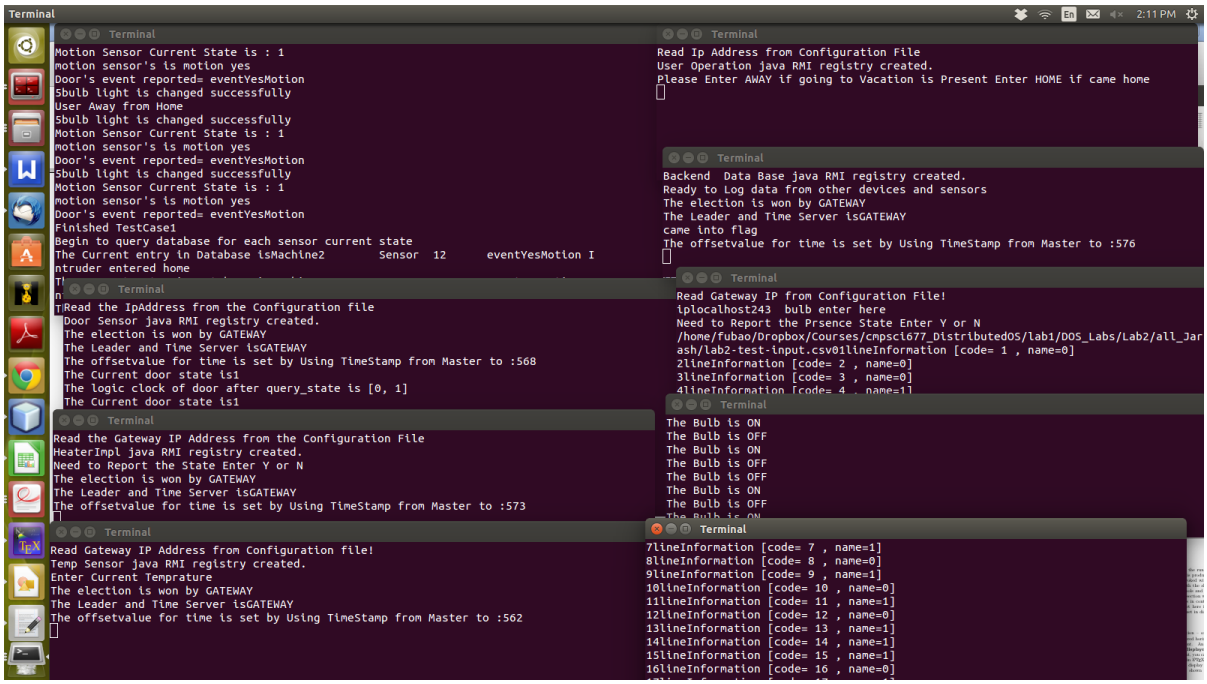
## Appendix 1: part1  Ring Algorithm




## Appendix 2: part2  Logical clock




## Appendix 3: test-cases , event ordering and automated part1 and part2




Note:

we run our test part1, part2 and part3 successfully in our ubuntu platform , without too much memory used, and we can get the right output result.
But when we run in the edlab, we have small problem sometimes, which is out of memor. Is it restricted by memory using in the edlab?