

Lab Report

This document is organized into 3 sections. In section 1, we discuss our implementation of ring algorithm for leader election which is an important part of Berkeley algorithm. Next, we discuss our implementation of totally ordered multicast which is used for event ordering based on logical Lamport clocks. Finally, we discuss our implementation of Berkeley algorithm for real clock synchronization.

In each section we start with a design overview, followed by test cases. Each test case is meant to cover use level functionality and component/algorithm correctness.

Section 1 – Leader election

Design

We follow the ring algorithm approach for leader election. Each time a node decides to start an election. It passes a message to its successor (next greater process ID). In the message it also appends its own Process ID. This way the message goes around in a ring till it finally comes back to the process who started the election. *At that time, the max amongst the process IDs is found and declared as the LEADER.* Once again a message is sent around the ring to declare the LEADER and notify the leader to take on the LEADER ship responsibility. This way leader election takes a constant time of $2(N-1)$ message passes. In our design we consider the failure of nodes as well. In this case, a process finds the next live node and passes the message to that instead of the immediate neighbor. This works recursively until either a successor is found or the given node is the only node in the ring at which time it is declared the LEADER. A snippet of the implementation is provided below. All the code concerning leader election is in the file *electionprocess.py* in the class *ElectionProcess* which every other device/sensor, gateway frontend inherit or subclass. To decide if a node is down or not we call a status method on the node to determine its status. Note: although we handle node failures, a failure of the frontend gateway is deemed critical as all functionality stops.

```
def getSucessor(self, prevId = None):
    nxtId = self.id + 1
    if prevId is not None:
        if prevId == self.id:
            self.nodestate = NodeECState.LEADER
            return self.id
        nxtId = prevId + 1

    num = len(self.pyro_ns.list()) - 3
    highestId = num + 2

    if nxtId > highestId:
        nxtId = 2

    if nxtId == 2:
        return "gateway"
    else:
        (ptype, pname) = Pyro4.Proxy("PYRONAME:gateway").getAddrFor(nxtId)
        try:
            isalive = Pyro4.Proxy("PYRONAME:" + ptype + "." + pname).getLiveStatus()
            if isalive:
                return (ptype, pname)
            else:
                return self.getSucessor(nxtId)
        except Exception:
            print("my sucessor {0} is down...trying next".format((ptype, pname)))
            return self.getSucessor(nxtId)
```

Note:

- Instead of providing all the host information of all the members (which is needed for finding the successor) we rely on the frontend gateway. This can be easily swapped to rely on the Pyro nameserver or can be passed explicitly for instance in the constructor during a process initialization

Tests

a) “all nodes live” leader election

This test can be simulated from the file `standalone_election_test.py`. It chooses a user specified node to begin the election algorithm. Ideally just like we used a time daemon in our Berkeley clock synchronization algorithm, we use an election daemon which runs leader election every few seconds. But for the sake of this test case and to track the log we show a standalone leader election as well.

```
def startelection():  
    Pyro4.Proxy("PYRONAME:Sensor.Door").start_election()
```

In this case the election is initiated by the door sensor. Following is the log of the *door sensor*, observe it initiates the ELECTION message which goes around the ring. Followed by which it sends the LEADER message choosing the max process ID 7 as the leader.

```
Z:\UMass\DOS\code\2ndlab\spring17-lab2-hsgodhia>python ./src/door_sensor.py localhost:9090  
Id 3  
init real clock value: 46  
Sensor: Door running with below uri:  
PYRO:Sensor.Door@localhost:51669  
starting election at 3  
got message (id:None,MsgType.ELECTION,data: [3, 4, 5, 6, 7, 2], lts: None, realts: None, from:2)  
got sucessor ('Sensor', 'Motion')  
got message (id:7,MsgType.LEADER,data: [3, 4, 5, 6, 7, 2], lts: None, realts: None, from:3)  
election complete
```

In this test case, process ID 7 corresponds to the outlet device. We view its logs next,

```
Z:\UMass\DOS\code\2ndlab\spring17-lab2-hsgodhia>python ./src/outlet_device.py localhost:9090  
init real clock value: 58  
Id 7  
Device: Outlet assigned id: 7 running with below uri.  
PYRO:Device.Outlet@localhost:51706  
got message (id:None,MsgType.ELECTION,data: [3, 4, 5, 6], lts: None, realts: None, from:6)  
got sucessor gateway  
got message (id:7,MsgType.LEADER,data: [3, 4, 5, 6], lts: None, realts: None, from:3)  
i am the leader  
leader is:7  
got sucessor gateway
```

Observe upon receiving a notification that it is the LEADER, it assumes the leader ship responsibility while the other nodes are set to be PARTICIPANT

```
if self.leaderid == self.id:  
    self.nodestate = NodeECState.LEADER  
    print("i am the leader")  
else:  
    self.nodestate = NodeECState.PARTICIPANT
```

To view the successor selection and message passing in action we also show the logs of an intermediate node (motion sensor)

```
Z:\UMass\DOS\code\2ndlab\spring17-lab2-hsgodhia>python ./src/motion_sensor.py localhost:9090
Id 4
init real clock value: 46
Sensor: Sensor running with below uri:
PYRO:Sensor.Motion@localhost:51668
got message (id:None,MsgType.ELECTION,data: [3], lts: None, realts: None, from:3)
got sucssor ('Sensor', 'Temperature')
got message (id:7,MsgType.LEADER,data: [3], lts: None, realts: None, from:3)
leader is:7
got sucssor ('Sensor', 'Temperature')
```

Performance (benchmark)

With all 5 sensor, devices and gateway running, this process takes 149 seconds to complete, elect leader

```
Z:\UMass\DOS\code\2ndlab\spring17-lab2-hsgodhia>python ./test/standalone_election_test.py
Starting simulation at 1491003885.4395287
End simulation at 1491004034.6491513
Durationi 149.20962262153625
```

b) “some nodes down” leader election

We now look at a test case when two nodes are down, this easily scales to more number of node failures, but for brevity and to not clutter the logs we consider this case. Consider the process IDs 4 (motion sensor) and 6 (bulb device) to be down. To make them down we simply stop the process CTRL + C. Like the previous test case we make the door sensor to initiate the election algorithm, its logs

```
starting election at 3
my sucssor ('Sensor', 'Motion') is down...trying next
got sucssor ('Sensor', 'Temperature')
got message (id:None,MsgType.ELECTION,data: [3, 5, 7, 2], lts: None, realts: None, from:2)
my sucssor ('Sensor', 'Motion') is down...trying next
got sucssor ('Sensor', 'Temperature')
got message (id:7,MsgType.LEADER,data: [3, 5, 7, 2], lts: None, realts: None, from:3)
election complete
```

We can observe from the logs of the door sensor(above) how it dealt with the node process ID 4 being down. It chose the next successor (process ID 5) and sent it the message. Upon receiving the ELECTION message back, observe that it doesn't contain the process IDs 4 and 6 which were down. From that list it choose the LEADER process ID 7 and again sends a message forward. Finally, upon receiving the LEADER message, with its populated list of process IDs including itself. It concludes the leader election.

From the logs we conclude as expected, the highest process ID 7(outlet) is elected the leader

```
Z:\UMass\DOS\code\2ndlab\spring17-lab2-hsgodhia>python ./src/outlet_device.py localhost:9090
init real clock value: 22
Id 7
Device: Outlet assigned id: 7 running with below uri.
PYRO:Device.Outlet@localhost:53708
got message (id:None,MsgType.ELECTION,data: [3, 5], lts: None, realts: None, from:5)
got sucssor gateway
got message (id:7,MsgType.LEADER,data: [3, 5], lts: None, realts: None, from:3)
i am the leader
leader is:7
got sucssor gateway
```

We look at the logs of another intermediate process (note how it skips over downed process ID 6)

```
Z:\UMass\DOS\code\2ndlab\spring17-lab2-hsgodhia>python ./src/temperature_sensor.py localhost:9090
init real clock value: 18
Id 5
Sensor: Temperature running with below uri:
PYRO:Sensor.Temperature@localhost:53697
got message (id:None,MsgType.ELECTION,data: [3], lts: None, realts: None, from:3)
my sucssor ('Device', 'Bulb') is down...trying next
got sucssor ('Device', 'Outlet')
got message (id:7,MsgType.LEADER,data: [3], lts: None, realts: None, from:3)
leader is:7
my sucssor ('Device', 'Bulb') is down...trying next
got sucssor ('Device', 'Outlet')
```

Note from this test we conclude—

1. Although some intermediate nodes were down, the ring leader election algorithm worked as expected and reported the max process ID 7 as the leader

Performance (benchmark)

```
Z:\UMass\DOS\code\2ndlab\spring17-lab2-hsgodhia>python ./test/standalone_election_test.py
Starting simulation at 1491008021.0905201
End simulation at 1491008168.2587328
Durationi 147.16821265220642
```

Section 2 – Logical Clock Synchronization

Design

Logical clock definition

We define the logical clock of each process as a simple integer counter that starts at 0 (can be changed to be a random integer as well) and increments by a fixed step/interval. Currently, each process logical clock increments by their $ID \% 2 + 1$. We keep this so that we can ensure that each process has a clock that increases at different speeds (even ID processes in steps of 1) odd ones ID in steps of 2. This makes the clock synchronization testing more rigorous and a good test bed for testing totally ordered multicast based on Lamport clock adjustments. It also matches the real world where machines have different crystal (time keeping stone) which results in machines ticking by different amounts and correspondingly difference in time. These clocks increment whenever a new event is SEND, RCVD, DELIVERED to the app layer. Note: logical clock has no relation with the real time

Totally ordered multicast

To ensure that all sensors, devices and frontend gateway receive the events in the same consistent order we implement *totally ordered multicast*.

Since totally ordered multicast assumes that messages from the same sender are received in the order of the send we first had to design code to account for this assumption. We implement this as—

1. Maintain an array of next expected message ids from each device/sensor/frontend gateway
2. When a process A sends a message to process B it includes a local message id to the message. Upon the receipt of the message at B, B looks up the array defined in 1) the next expected message id with respect to A. If the current message matches this message id, it is processed

ahead else it is buffered. *Below is a snippet describing the implementation.* Note: Although the code shows a while 1 loop, this is highly efficient since the queue upon which we call *get* is a blocking call that blocks until a new event or message has arrived. So the while loop doesn't do a busy waiting and hence it is efficient

```
def rcvd_msg_queue_reader(self):
    while 1:
        msg = self.rcvmsg_q.get()
        if msg.msgid == self.msghist.get(msg.frompid, 0) + 1:
            self.msghist[msg.frompid] = self.msghist.get(msg.frompid, 0) + 1
            self.deliver(msg)
        else:
            self.rcvmsg_q.put(msg)
```

Now, for logical clock synchronization the algorithm followed is—

1. When sending a message from process A to process B. In addition to sending the message to process B, *multicast* the message to all the members in the group (in our case all the devices, sensors and frontend gateway). Note: we don't make the backend gateway a part of the multicast group as only the gateway frontend is privileged to communicate with the backend
2. Upon a receipt of a message at a node X, the message is buffered in a local queue at X. In addition an ACK for that message is sent to all other N-1 members of the multicast group.
3. A message is buffered in the local queue at X until it has received an ACK from the other N-1 members of the multicast group.
4. The queue is implemented as a blocking priority queue using the “queue” package of python 3. The priority of a message in the queue is defined by its logical time stamp (*we break ties* in the logical clock time stamp by assuming the message from a lower process id to be considered before the one from the higher process id). We implement Lamport's algorithm for clock adjustments:
 - a. Set the logical clock value to the maximum of the message time stamp and current logical clock value)
 - b. Increment the clock value

This ensures totally ordered multicast, a snippet of the code is below. To ensure clean code we abstract all the methods and implementations of the logical clock in a class LogicalProcess in the file *logicalprocess.py* which every sensor, device and frontend gateway inherit from.

```
def deliver(self, msg):
    #print("got message {}".format(msg), flush=True)
    if msg.msg_type == MessageType.DATA or msg.msg_type == MessageType.STATE:
        self.msg_queue.put(msg)
        self.setlogclock(max(self.logicalclock, msg.ts_logical) + 1)
        print("logical ts {} RCVD {}".format(self.logicalclock, msg))
        nmsg = Message(id = self.getNextMsgId(), pid = self.id, data = (msg.msgid, msg.frompid), msg_type=msg.msg_type)
        self.multicast(nmsg)

    elif msg.msg_type == MessageType.ACK:
        global _msgack_lock
        with _msgack_lock:
            curval = self.msg_acks.get(msg.data, 0)
            self.msg_acks[msg.data] = curval + 1
```

Test

a) Case “series of temperature changes”

Here, we simulate a series of temperature changes. In this test case we are testing to ensure that messages from the same sender (temperature sensor) are received in the same order at each of the other devices/sensors and gateway frontend and DB text file. This is demonstrated in the *mult_msg.py* file in the test directory.

```
t1 = threading.Thread(target = tempChange, args=("34F",))
t2 = threading.Thread(target = tempChange, args=("40F",))
t3 = threading.Thread(target = tempChange, args=("29F",))
t4 = threading.Thread(target = tempChange, args=("60F",))
t5 = threading.Thread(target = tempChange, args=("61F",))
t6 = threading.Thread(target = tempChange, args=("45F",))
t7 = threading.Thread(target = tempChange, args=("41F",))
|
t1.start()
t2.start()
t3.start()
t4.start()
t5.start()
t6.start()
t7.start()
```

Note: since we are using threads to simulate the temperature change, the above temperature changes are initiated in *near parallel and since they get interleaved* we cannot say what the final temperature is but we can say for sure that all devices, sensors and frontend gateway follow the same sequence of temperature changes and reach the same final temperature because we implement *totally ordered multicast*. This is confirmed by the program *mult_msg.py* and the logs. For conciseness we show the log of the frontend gateway, one device and one sensor.

Analysis of logs—

- All components show the DELIVERED event/messages in the same order which displays the successful working of totally ordered multicast

Frontend gateway

```
Clock mode set to LOGICAL
logical ts 24 RCVD (id:1,MsgType.STATE,data: 41F, lts: 23, realts: 52, from:7)
logical ts 26 RCVD (id:2,MsgType.STATE,data: 61F, lts: 25, realts: 60, from:7)
logical ts 28 RCVD (id:3,MsgType.STATE,data: 60F, lts: 27, realts: 68, from:7)
logical ts 30 RCVD (id:4,MsgType.STATE,data: 34F, lts: 29, realts: 76, from:7)
logical ts 32 RCVD (id:5,MsgType.STATE,data: 40F, lts: 31, realts: 84, from:7)
logical ts 34 RCVD (id:6,MsgType.STATE,data: 45F, lts: 33, realts: 92, from:7)
logical ts 36 RCVD (id:7,MsgType.STATE,data: 29F, lts: 35, realts: 100, from:7)
ts-logical: 37 ts-real: 96 DELIVERED (id:1,MsgType.STATE,data: 41F, lts: 23, realts: 52, from:7)
ts-logical: 38 ts-real: 99 DELIVERED (id:2,MsgType.STATE,data: 61F, lts: 25, realts: 60, from:7)
ts-logical: 39 ts-real: 102 DELIVERED (id:3,MsgType.STATE,data: 60F, lts: 27, realts: 68, from:7)
ts-logical: 40 ts-real: 105 DELIVERED (id:4,MsgType.STATE,data: 34F, lts: 29, realts: 76, from:7)
ts-logical: 41 ts-real: 108 DELIVERED (id:5,MsgType.STATE,data: 40F, lts: 31, realts: 84, from:7)
ts-logical: 42 ts-real: 111 DELIVERED (id:6,MsgType.STATE,data: 45F, lts: 33, realts: 92, from:7)
ts-logical: 43 ts-real: 114 DELIVERED (id:7,MsgType.STATE,data: 29F, lts: 35, realts: 100, from:7)
```


Door sensor

```
Clock mode set to LOGICAL
logical ts 24 RCVD (id:1,MsgType.STATE,data: 41F, lts: 23, realts: 52, from:7)
logical ts 26 RCVD (id:2,MsgType.STATE,data: 61F, lts: 25, realts: 60, from:7)
logical ts 28 RCVD (id:3,MsgType.STATE,data: 60F, lts: 27, realts: 68, from:7)
logical ts 30 RCVD (id:4,MsgType.STATE,data: 34F, lts: 29, realts: 76, from:7)
logical ts 32 RCVD (id:5,MsgType.STATE,data: 40F, lts: 31, realts: 84, from:7)
logical ts 34 RCVD (id:6,MsgType.STATE,data: 45F, lts: 33, realts: 92, from:7)
logical ts 36 RCVD (id:7,MsgType.STATE,data: 29F, lts: 35, realts: 100, from:7)
logical ts 38 real ts 128 DELIVERED (id:1,MsgType.STATE,data: 41F, lts: 23, realts: 52, from:7)
logical ts 40 real ts 132 DELIVERED (id:2,MsgType.STATE,data: 61F, lts: 25, realts: 60, from:7)
logical ts 42 real ts 136 DELIVERED (id:3,MsgType.STATE,data: 60F, lts: 27, realts: 68, from:7)
logical ts 44 real ts 140 DELIVERED (id:4,MsgType.STATE,data: 34F, lts: 29, realts: 76, from:7)
logical ts 46 real ts 144 DELIVERED (id:5,MsgType.STATE,data: 40F, lts: 31, realts: 84, from:7)
logical ts 48 real ts 148 DELIVERED (id:6,MsgType.STATE,data: 45F, lts: 33, realts: 92, from:7)
logical ts 50 real ts 152 DELIVERED (id:7,MsgType.STATE,data: 29F, lts: 35, realts: 100, from:7)
```

Bulb device

```
Clock mode set to LOGICAL
logical ts 24 RCVD (id:1,MsgType.STATE,data: 41F, lts: 23, realts: 52, from:7)
logical ts 26 RCVD (id:2,MsgType.STATE,data: 61F, lts: 25, realts: 60, from:7)
logical ts 28 RCVD (id:3,MsgType.STATE,data: 60F, lts: 27, realts: 68, from:7)
logical ts 30 RCVD (id:4,MsgType.STATE,data: 34F, lts: 29, realts: 76, from:7)
logical ts 32 RCVD (id:5,MsgType.STATE,data: 40F, lts: 31, realts: 84, from:7)
logical ts 34 RCVD (id:6,MsgType.STATE,data: 45F, lts: 33, realts: 92, from:7)
logical ts 36 RCVD (id:7,MsgType.STATE,data: 29F, lts: 35, realts: 100, from:7)
ts-logical: 38 ts-real: 128 DELIVERED (id:1,MsgType.STATE,data: 41F, lts: 23, realts: 52, from:7)
ts-logical: 40 ts-real: 134 DELIVERED (id:2,MsgType.STATE,data: 61F, lts: 25, realts: 60, from:7)
ts-logical: 42 ts-real: 140 DELIVERED (id:3,MsgType.STATE,data: 60F, lts: 27, realts: 68, from:7)
ts-logical: 44 ts-real: 146 DELIVERED (id:4,MsgType.STATE,data: 34F, lts: 29, realts: 76, from:7)
ts-logical: 46 ts-real: 152 DELIVERED (id:5,MsgType.STATE,data: 40F, lts: 31, realts: 84, from:7)
ts-logical: 48 ts-real: 158 DELIVERED (id:6,MsgType.STATE,data: 45F, lts: 33, realts: 92, from:7)
ts-logical: 50 ts-real: 164 DELIVERED (id:7,MsgType.STATE,data: 29F, lts: 35, realts: 100, from:7)
```

DB

```
Motion:NO Door:CLOSED Temperature:41F Bulb:NA Outlet:NA logical_clock:37 real_clock:96
Motion:NO Door:CLOSED Temperature:61F Bulb:NA Outlet:NA logical_clock:38 real_clock:99
Motion:NO Door:CLOSED Temperature:60F Bulb:NA Outlet:NA logical_clock:39 real_clock:102
Motion:NO Door:CLOSED Temperature:34F Bulb:NA Outlet:NA logical_clock:40 real_clock:105
Motion:NO Door:CLOSED Temperature:40F Bulb:NA Outlet:NA logical_clock:41 real_clock:108
Motion:NO Door:CLOSED Temperature:45F Bulb:NA Outlet:NA logical_clock:42 real_clock:111
Motion:NO Door:CLOSED Temperature:29F Bulb:NA Outlet:NA logical_clock:43 real_clock:114
```

b) Case “user enters house”

Consider the test case of a user entering the house. The sequence of events triggered by the devices are simulated in the *user_enter_exit.py* file. This file thus also acts as our user process which triggers the motion sense, door open etc. The following code snippet indicates this. Note: instead of assuming the keychain (or beacon) as a separate sensor we assume that the door sensor itself has a beacon or hardware which sends an additional KEY event along with the OPEN event if the user enters using a keychain. If a thief tries to enter, it will ideally use a non-standard method to enter, in this case the door sensor does not send a KEY event

```
def user_enter():
    Pyro4.Proxy("PYRONAME:Sensor.Door").toggle_state("OPEN")
    Pyro4.Proxy("PYRONAME:Sensor.Door").toggle_state("KEY")
    Pyro4.Proxy("PYRONAME:Sensor.Motion").toggle_state("YES")
    Pyro4.Proxy("PYRONAME:Sensor.Door").toggle_state("CLOSED")
    Pyro4.Proxy("PYRONAME:Sensor.Motion").toggle_state("NO")
```

Note: In the above snippet, toggling basically means we set the state of the device to the parameter passed. Since Door and Motion are Push based sensors as soon as the above command runs in their process it triggers a push event. This test case is for logical clocks so in addition to sending the push event to the frontend gateway it is also sent to all the other sensors/devices in the system.

We now look at the logs of the DB (text file backend) and few components, the frontend gateway, door sensor and bulb device.

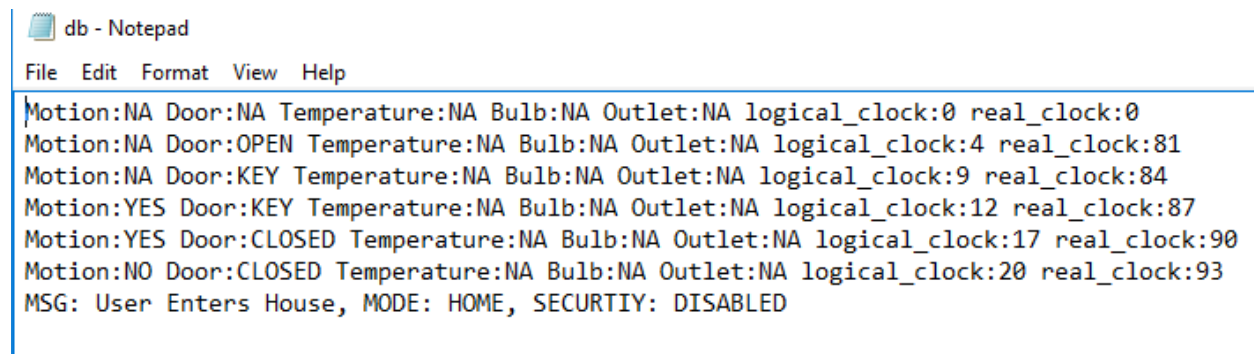
Analysis of logs—

Each event is described in BOLD face as RCVD (which indicates the first receipt of the event) and DELIVERED (which indicates the event has been delivered to the application layer). Each line also contains the message payload, the payload is modeled as an object of class Message in message.py. Each message is uniquely identified by a composite key made up of the process id of the sender (**from**) and unique message id (**id**). Upon receipt of a message we follow Lamport algorithm to adjust the clock: it is set to the **max-of** (the message logical time stamp (**Its**), own logical clock value) + 1. To better visualize this sequence of events at the gateway we plotted it. The file plot.py will generate the plots (it doesn't automatically parse the above logs, the x and y axis values are entered as above log)

Looking at the sequence of the events, the DB screenshot and the plot we conclude 3 things—

1. Each device, bulb and frontend gateway receive the events in the same order and Lamport clock adjustment with totally ordered multicast is successful in ordering the events
2. The event sequence at the frontend gateway is the source of truth for enabling or disabling the security system of the home. In this case the frontend gateway successfully identifies a sequence of events which match that of a user entering the house and hence would disable the security system
3. The System correctly captures the sequence of events and logs in the DB that user is entered, house mode is HOME and security is DISABLED

DB



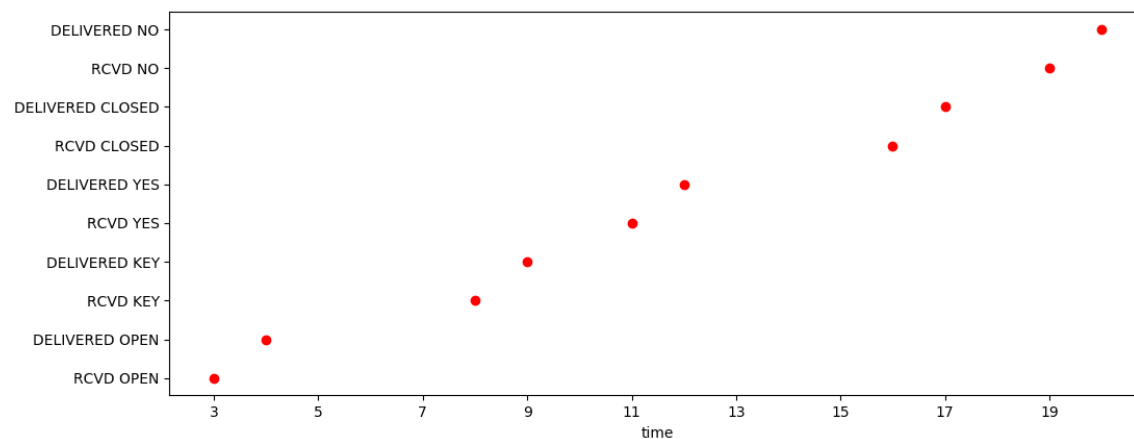
```
db - Notepad
File Edit Format View Help
Motion:NA Door:NA Temperature:NA Bulb:NA Outlet:NA logical_clock:0 real_clock:0
Motion:NA Door:OPEN Temperature:NA Bulb:NA Outlet:NA logical_clock:4 real_clock:81
Motion:NA Door:KEY Temperature:NA Bulb:NA Outlet:NA logical_clock:9 real_clock:84
Motion:YES Door:KEY Temperature:NA Bulb:NA Outlet:NA logical_clock:12 real_clock:87
Motion:YES Door:CLOSED Temperature:NA Bulb:NA Outlet:NA logical_clock:17 real_clock:90
Motion:NO Door:CLOSED Temperature:NA Bulb:NA Outlet:NA logical_clock:20 real_clock:93
MSG: User Enters House, MODE: HOME, SECURITY: DISABLED
```

Gateway frontend (log and plot of sequence of events)


```

Z:\UMass\DOS\code\2ndlab\spring17-lab2-hsgodhia>python ./src/gatewayfront.py localhost:9090
gateway running with below uri:
init real clock value: 78
Id 2
PYRO:gateway@localhost:61174
Clock mode set to LOGICAL
logical ts 3 RCVD (id:1,MsgType.STATE,data: OPEN, lts: 2, realts: 96, from:3)
ts-logical: 4 ts-real: 81 DELIVERED (id:1,MsgType.STATE,data: OPEN, lts: 2, realts: 96, from:3)
logical ts 8 RCVD (id:3,MsgType.STATE,data: KEY, lts: 7, realts: 104, from:3)
ts-logical: 9 ts-real: 84 DELIVERED (id:3,MsgType.STATE,data: KEY, lts: 7, realts: 104, from:3)
logical ts 11 RCVD (id:3,MsgType.STATE,data: YES, lts: 10, realts: 108, from:4)
ts-logical: 12 ts-real: 87 DELIVERED (id:3,MsgType.STATE,data: YES, lts: 10, realts: 108, from:4)
Gateway Remotely switching ON bulb
logical ts 16 RCVD (id:6,MsgType.STATE,data: CLOSED, lts: 15, realts: 116, from:3)
ts-logical: 17 ts-real: 90 DELIVERED (id:6,MsgType.STATE,data: CLOSED, lts: 15, realts: 116, from:3)
logical ts 19 RCVD (id:6,MsgType.STATE,data: NO, lts: 18, realts: 123, from:4)
ts-logical: 20 ts-real: 93 DELIVERED (id:6,MsgType.STATE,data: NO, lts: 18, realts: 123, from:4)
Gateway Remotely switching OFF bulb

```



We further show the logs for open sensor and device, as the remaining would be analogous

Door sensor

```

Z:\UMass\DOS\code\2ndlab\spring17-lab2-hsgodhia>python ./src/door_sensor.py localhost:9090
Id 3
init real clock value: 92
Sensor: Door running with below uri:
PYRO:Sensor.Door@localhost:61181
Clock mode set to LOGICAL
logical ts 2 SEND event (id:1,MsgType.STATE,data: OPEN, lts: 2, realts: 96, from:3)
logical ts 3 RCVD (id:1,MsgType.STATE,data: OPEN, lts: 2, realts: 96, from:3)
logical ts 5 real ts 100 DELIVERED (id:1,MsgType.STATE,data: OPEN, lts: 2, realts: 96, from:3)
logical ts 7 SEND event (id:3,MsgType.STATE,data: KEY, lts: 7, realts: 104, from:3)
logical ts 8 RCVD (id:3,MsgType.STATE,data: KEY, lts: 7, realts: 104, from:3)
logical ts 10 real ts 108 DELIVERED (id:3,MsgType.STATE,data: KEY, lts: 7, realts: 104, from:3)
logical ts 11 RCVD (id:3,MsgType.STATE,data: YES, lts: 10, realts: 108, from:4)
logical ts 13 real ts 112 DELIVERED (id:3,MsgType.STATE,data: YES, lts: 10, realts: 108, from:4)
logical ts 15 SEND event (id:6,MsgType.STATE,data: CLOSED, lts: 15, realts: 116, from:3)
logical ts 16 RCVD (id:6,MsgType.STATE,data: CLOSED, lts: 15, realts: 116, from:3)
logical ts 18 real ts 120 DELIVERED (id:6,MsgType.STATE,data: CLOSED, lts: 15, realts: 116, from:3)
logical ts 19 RCVD (id:6,MsgType.STATE,data: NO, lts: 18, realts: 123, from:4)
logical ts 21 real ts 124 DELIVERED (id:6,MsgType.STATE,data: NO, lts: 18, realts: 123, from:4)

```

Bulb device

```

Z:\UMass\DOS\code\2ndlab\spring17-lab2-hsgodhia>python ./src/bulb_device.py localhost:9090
init real clock value: 92
Id 5
Device: Bulb assigned id: 5 running with below uri.
PYRO:Device.Bulb@localhost:61192
Clock mode set to LOGICAL
logical ts 3 RCVD (id:1,MsgType.STATE,data: OPEN, lts: 2, realts: 96, from:3)
ts-logical: 5 ts-real: 98 DELIVERED (id:1,MsgType.STATE,data: OPEN, lts: 2, realts: 96, from:3)
logical ts 8 RCVD (id:3,MsgType.STATE,data: KEY, lts: 7, realts: 104, from:3)
ts-logical: 10 ts-real: 104 DELIVERED (id:3,MsgType.STATE,data: KEY, lts: 7, realts: 104, from:3)
logical ts 11 RCVD (id:3,MsgType.STATE,data: YES, lts: 10, realts: 108, from:4)
ts-logical: 13 ts-real: 110 DELIVERED (id:3,MsgType.STATE,data: YES, lts: 10, realts: 108, from:4)
logical ts 16 RCVD (id:6,MsgType.STATE,data: CLOSED, lts: 15, realts: 116, from:3)
Bulb is switched: ON
ts-logical: 18 ts-real: 116 DELIVERED (id:6,MsgType.STATE,data: CLOSED, lts: 15, realts: 116, from:3)
logical ts 19 RCVD (id:6,MsgType.STATE,data: NO, lts: 18, realts: 123, from:4)
ts-logical: 21 ts-real: 122 DELIVERED (id:6,MsgType.STATE,data: NO, lts: 18, realts: 123, from:4)
Bulb is switched: OFF

```

Performance benchmark

We observed the time taken for this operation to complete is **22s**

```

Z:\UMass\DOS\code\2ndlab\spring17-lab2-hsgodhia>python ./test/user_enter_exit.py
Preparing and initializing...
Starting simulation at 1490977420.077363
End simulation at 1490977442.7329867
Duration: 22.6556236743927

```

c) Case “bulb Lights on”

In this test case we simulate an automatic bulb on and off based on the events sent by the motion sensor. This simulation can be run using the file *lights_on.py*. Below is an excerpt of the log of the **motion sensor** during this scenario. We notice the SEND event with state NO at time 21 (ts logical), which is DELIVERED to the app layer at time 23. Similarly, we have SEND event with state YES at time 24 and NO at time 27 each of which is DELIVERED to the app layer at time 26 and 29 respectively.

```

Clock mode set to LOGICAL
logical ts 21 SEND event (id:1,MsgType.STATE,data: NO, lts: 21, realts: 139, from:4)
logical ts 22 RCVD (id:1,MsgType.STATE,data: NO, lts: 21, realts: 139, from:4)
ts-logical: 23 ts-real: 144 DELIVERED (id:1,MsgType.STATE,data: NO, lts: 21, realts: 139, from:4)
logical ts 24 SEND event (id:3,MsgType.STATE,data: YES, lts: 24, realts: 149, from:4)
logical ts 25 RCVD (id:3,MsgType.STATE,data: YES, lts: 24, realts: 149, from:4)
ts-logical: 26 ts-real: 154 DELIVERED (id:3,MsgType.STATE,data: YES, lts: 24, realts: 149, from:4)
logical ts 27 SEND event (id:5,MsgType.STATE,data: NO, lts: 27, realts: 159, from:4)
logical ts 28 RCVD (id:5,MsgType.STATE,data: NO, lts: 27, realts: 159, from:4)
ts-logical: 29 ts-real: 164 DELIVERED (id:5,MsgType.STATE,data: NO, lts: 27, realts: 159, from:4)

```

Next, below is an excerpt of the logs of **bulb device** shows how it receives the events messages. From looking at the sequence of events we can conclude that the bulb receives the events in correct expected order of NO, YES, NO. When the event received is YES the bulb is automatically switched on and when it again receives NO it is switched off back.

```

Clock mode set to LOGICAL
logical ts 22 RCVD (id:1,MsgType.STATE,data: NO, lts: 21, realts: 139, from:4)
ts-logical: 24 ts-real: 37 DELIVERED (id:1,MsgType.STATE,data: NO, lts: 21, realts: 139, from:4)
logical ts 25 RCVD (id:3,MsgType.STATE,data: YES, lts: 24, realts: 149, from:4)
ts-logical: 27 ts-real: 43 DELIVERED (id:3,MsgType.STATE,data: YES, lts: 24, realts: 149, from:4)
Bulb is switched: OFF
Bulb is switched: ON
logical ts 28 RCVD (id:5,MsgType.STATE,data: NO, lts: 27, realts: 159, from:4)
ts-logical: 30 ts-real: 49 DELIVERED (id:5,MsgType.STATE,data: NO, lts: 27, realts: 159, from:4)
Bulb is switched: OFF

```

Next, below is an excerpt of the log of the frontend gateway

```

Clock mode set to LOGICAL
logical ts 22 RCVD (id:1,MsgType.STATE,data: NO, lts: 21, realts: 139, from:4)
ts-logical: 23 ts-real: 102 DELIVERED (id:1,MsgType.STATE,data: NO, lts: 21, realts: 139, from:4)
Gateway Remotely switching OFF bulb
logical ts 25 RCVD (id:3,MsgType.STATE,data: YES, lts: 24, realts: 149, from:4)
ts-logical: 26 ts-real: 105 DELIVERED (id:3,MsgType.STATE,data: YES, lts: 24, realts: 149, from:4)
Gateway Remotely switching ON bulb
logical ts 28 RCVD (id:5,MsgType.STATE,data: NO, lts: 27, realts: 159, from:4)
ts-logical: 29 ts-real: 108 DELIVERED (id:5,MsgType.STATE,data: NO, lts: 27, realts: 159, from:4)
Gateway Remotely switching OFF bulb

```

The key point to observe in this test case is—

- The motion sensor detects motion, and SEND its event to the frontend gateway which REMOTELY switches ON/OFF the bulb
- This test case shows the working of remote control and event ordering combined

Performance (benchmark numbers)

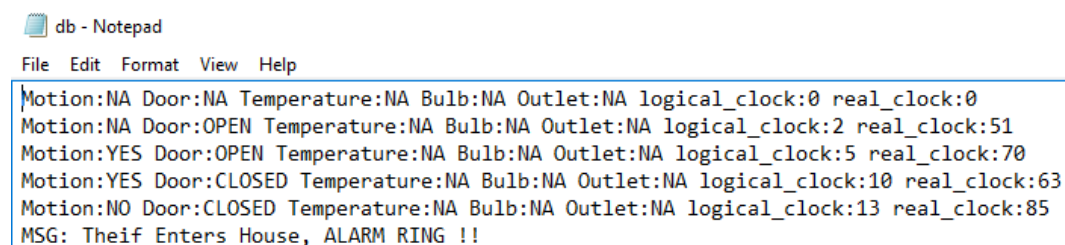
```

Z:\UMass\DOS\code\2ndlab\spring17-lab2-hsgodhia>python ./test/lights_on.py
Preparing and initializing...
Starting simulation at 1490978157.6914392
End simulation at 1490978174.2822242
Duration 16.590785026550293

```

d) Thief enters house

This test case can be found in the file *thief_enter.py*. It simulates a thief breaking into the house. Without the essential “key” beacon the events sent by the motion and door sensor help the system determine a break in. This is demonstrated in the DB logs which print out a break in and raise the ALARM. It also demonstrates the functionality of the gateway to differentiate between sequence of events and accordingly raise and appropriate message/alarm



```

db - Notepad
File Edit Format View Help
Motion:NA Door:NA Temperature:NA Bulb:NA Outlet:NA logical_clock:0 real_clock:0
Motion:NA Door:OPEN Temperature:NA Bulb:NA Outlet:NA logical_clock:2 real_clock:51
Motion:YES Door:OPEN Temperature:NA Bulb:NA Outlet:NA logical_clock:5 real_clock:70
Motion:YES Door:CLOSED Temperature:NA Bulb:NA Outlet:NA logical_clock:10 real_clock:63
Motion:NO Door:CLOSED Temperature:NA Bulb:NA Outlet:NA logical_clock:13 real_clock:85
MSG: Thief Enters House, ALARM RING !!

```

The sequence of events at the happening at the frontend gateway are

```

Z:\UMass\DOS\code\2ndlab\spring17-lab2-hsgodhia>python ./src/gatewayfront.py localhost:9090
gateway running with below uri:
init real clock value: 50
Id 2
PYRO:gateway@localhost:49325
Clock mode set to LOGICAL
Clock mode set to LOGICAL
logical ts 3 RCVD (id:1,MsgType.STATE,data: OPEN, lts: 2, realts: 51, from:3)
ts-logical: 4 ts-real: 53 DELIVERED (id:1,MsgType.STATE,data: OPEN, lts: 2, realts: 51, from:3)
logical ts 6 RCVD (id:2,MsgType.STATE,data: YES, lts: 5, realts: 70, from:4)
ts-logical: 7 ts-real: 56 DELIVERED (id:2,MsgType.STATE,data: YES, lts: 5, realts: 70, from:4)
Gateway Remotely switching ON bulb
logical ts 11 RCVD (id:4,MsgType.STATE,data: CLOSED, lts: 10, realts: 63, from:3)
ts-logical: 12 ts-real: 59 DELIVERED (id:4,MsgType.STATE,data: CLOSED, lts: 10, realts: 63, from:3)
logical ts 14 RCVD (id:5,MsgType.STATE,data: NO, lts: 13, realts: 85, from:4)
ts-logical: 15 ts-real: 62 DELIVERED (id:5,MsgType.STATE,data: NO, lts: 13, realts: 85, from:4)
Gateway Remotely switching OFF bulb

```

The sequence of events at the door and motion sensor are,

```

Z:\UMass\DOS\code\2ndlab\spring17-lab2-hsgodhia>python ./src/door_sensor.py localhost:9090
Id 3
init real clock value: 47
Sensor: Door running with below uri:
PYRO:Sensor.Door@localhost:49339
Clock mode set to LOGICAL
Clock mode set to LOGICAL
logical ts 2 SEND event (id:1,MsgType.STATE,data: OPEN, lts: 2, realts: 51, from:3)
logical ts 3 RCVD (id:1,MsgType.STATE,data: OPEN, lts: 2, realts: 51, from:3)
logical ts 5 real ts 55 DELIVERED (id:1,MsgType.STATE,data: OPEN, lts: 2, realts: 51, from:3)
logical ts 6 RCVD (id:2,MsgType.STATE,data: YES, lts: 5, realts: 70, from:4)
logical ts 8 real ts 59 DELIVERED (id:2,MsgType.STATE,data: YES, lts: 5, realts: 70, from:4)
logical ts 10 SEND event (id:4,MsgType.STATE,data: CLOSED, lts: 10, realts: 63, from:3)
logical ts 11 RCVD (id:4,MsgType.STATE,data: CLOSED, lts: 10, realts: 63, from:3)
logical ts 13 real ts 67 DELIVERED (id:4,MsgType.STATE,data: CLOSED, lts: 10, realts: 63, from:3)
logical ts 14 RCVD (id:5,MsgType.STATE,data: NO, lts: 13, realts: 85, from:4)
logical ts 16 real ts 71 DELIVERED (id:5,MsgType.STATE,data: NO, lts: 13, realts: 85, from:4)

```

```

Z:\UMass\DOS\code\2ndlab\spring17-lab2-hsgodhia>python ./src/motion_sensor.py localhost:9090
Id 4
init real clock value: 60
Sensor: Sensor running with below uri:
PYRO:Sensor.Motion@localhost:49344
Clock mode set to LOGICAL
Clock mode set to LOGICAL
logical ts 3 RCVD (id:1,MsgType.STATE,data: OPEN, lts: 2, realts: 51, from:3)
ts-logical: 4 ts-real: 65 DELIVERED (id:1,MsgType.STATE,data: OPEN, lts: 2, realts: 51, from:3)
logical ts 5 SEND event (id:2,MsgType.STATE,data: YES, lts: 5, realts: 70, from:4)
logical ts 6 RCVD (id:2,MsgType.STATE,data: YES, lts: 5, realts: 70, from:4)
ts-logical: 7 ts-real: 75 DELIVERED (id:2,MsgType.STATE,data: YES, lts: 5, realts: 70, from:4)
logical ts 11 RCVD (id:4,MsgType.STATE,data: CLOSED, lts: 10, realts: 63, from:3)
ts-logical: 12 ts-real: 80 DELIVERED (id:4,MsgType.STATE,data: CLOSED, lts: 10, realts: 63, from:3)
logical ts 13 SEND event (id:5,MsgType.STATE,data: NO, lts: 13, realts: 85, from:4)
logical ts 14 RCVD (id:5,MsgType.STATE,data: NO, lts: 13, realts: 85, from:4)
ts-logical: 15 ts-real: 90 DELIVERED (id:5,MsgType.STATE,data: NO, lts: 13, realts: 85, from:4)

```

Section 3 – Berkeley Algorithm

Design

Real clock

Real clock definition. We model the real clock by taking the system time % 100. We do this to deal with small numbers. Further to ensure that different process have different time and correctly simulate the clocks we assume each process increments its own real clock by a different number. Currently it is set to

process PID but it can effectively be any random integer. We have also added a random initial offset. Given this accurate modeling of a real clock we know move to the design of the Berkeley algorithm.

Berkeley algorithm

We implement the mechanics of Berkeley algorithm in a class called *RealProcess* in the *realprocess.py* file. This is inherited by every sensor/device and frontend gateway giving them the capability to participate in the Berkeley algorithm. As a part of the inherited implementation, each process runs the following time daemon. If the process running the daemon detects that it is the leader (chosen based of the leader election algorithm) it initiates the Berkeley algorithm to sync the time by polling all the other devices and sensors finding the average time and pushing the average time to every other device/sensor so they may update their real clock value. The following code excerpt shows the initialization of the time daemon as a background process which continuously runs and the implementation of the Berkeley Sync algorithm.

Note:

- the backend DB doesn't participate in the leader election and Berkeley sync algorithm
- When computing the average time for Berkeley we consider the network delay that is incurred for each process and account that as well
- If a process is down and doesn't respond it is ignored when computing the average time

```
t = threading.Thread(target = self.timeDaemon)
t.setDaemon(True)
t.start()
```

```
avgttime = 0
for k in proxies:
    if k in down_machines:
        continue
    try:
        #only calling methods on dead proxies is a problem
        start_time = time.time()
        avgttime += proxies[k].getRealts()
        end_time = time.time()
        #we consider the network delay while computing the avg time as well
        delay = (end_time - start_time)/2
        avgttime += delay
    except Exception:
        #continous log is printed if a node is down, to remind user to start the node
        print("node {0} is down".format(members[k]), flush=True)
        down_machines.append(k)

avgttime = avgttime/len(proxies)
avgttime = round(avgttime)

for k in proxies:
    if k == "Pyro.NameServer" or k == "db":
        continue
    if k in down_machines:
        continue
    proxies[k].setRealts(avgttime)
```

Tests

- a) case "series of temperature changes"

Here, we simulate a series of temperature of changes. In case of real clock synchronization the temperature sensor directly pushes the events to the frontend gateway. There is no multicast involved as compared to the Lamport logical clocks synchronization. At the frontend gateway upon receiving messages it is inserted into a priority queue. The priority of an event/message received is determined by its real time stamp. Since Berkeley algorithm is configured to continuously run once every second we expect the real clocks to be synchronized and this ensures event ordering. Ideally a temperature sensor should send events of temperature changes spaced by some time. We consider test for two cases one where the time changes are sent sequentially and one where the time changes are sent in parallel concurrently (although this is hypothetical since there is a single temperature sensor and it would send out events sequentially).

- Real world case of Sequential temperature changes

The frontend gateway log

```
Clock mode set to REAL
ts-logical: 8 ts-real: 25 DELIVERED (id:1,MsgType.STATE,data: 34F, lts: 8, realts: 72, from:4)
ts-logical: 9 ts-real: 26 DELIVERED (id:2,MsgType.STATE,data: 40F, lts: 9, realts: 73, from:4)
ts-logical: 10 ts-real: 27 DELIVERED (id:3,MsgType.STATE,data: 29F, lts: 10, realts: 74, from:4)
ts-logical: 11 ts-real: 28 DELIVERED (id:4,MsgType.STATE,data: 60F, lts: 11, realts: 75, from:4)
ts-logical: 12 ts-real: 29 DELIVERED (id:5,MsgType.STATE,data: 61F, lts: 12, realts: 76, from:4)
ts-logical: 13 ts-real: 30 DELIVERED (id:6,MsgType.STATE,data: 45F, lts: 13, realts: 77, from:4)
ts-logical: 14 ts-real: 31 DELIVERED (id:7,MsgType.STATE,data: 41F, lts: 14, realts: 78, from:4)
```

We notice that the messages DELIVERED to the gateway are in the order of the real clock values or time stamps of the events/messages indicated by the field (**real ts**)

The temperature sensor log with the SEND events and the data payload (in the **data** field).

```
Clock mode set to REAL
real ts: 72 SEND event (id:1,MsgType.STATE,data: 34F, lts: 8, realts: 72, from:4)
real ts: 73 SEND event (id:2,MsgType.STATE,data: 40F, lts: 9, realts: 73, from:4)
real ts: 74 SEND event (id:3,MsgType.STATE,data: 29F, lts: 10, realts: 74, from:4)
real ts: 75 SEND event (id:4,MsgType.STATE,data: 60F, lts: 11, realts: 75, from:4)
real ts: 76 SEND event (id:5,MsgType.STATE,data: 61F, lts: 12, realts: 76, from:4)
real ts: 77 SEND event (id:6,MsgType.STATE,data: 45F, lts: 13, realts: 77, from:4)
real ts: 78 SEND event (id:7,MsgType.STATE,data: 41F, lts: 14, realts: 78, from:4)
```

Analysis of logs—

Notice that the events are received in the same order as the user input (thereby corresponding to the same sequence of temperature changes) and resulting in the same final temperature. Refer to the test file `seq_time_change_real.py` for more details

```
for temp in ["34F", "40F", "29F", "60F", "61F", "45F", "41F"]:
    tempChange(temp)
```


- Hypothetical case of concurrent temperature changes (changes in concurrent fashion)

```
t1 = threading.Thread(target = tempChange, args=("34F",))
t2 = threading.Thread(target = tempChange, args=("40F",))
t3 = threading.Thread(target = tempChange, args=("29F",))
t4 = threading.Thread(target = tempChange, args=("60F",))
t5 = threading.Thread(target = tempChange, args=("61F",))
t6 = threading.Thread(target = tempChange, args=("45F",))
t7 = threading.Thread(target = tempChange, args=("41F",))

t1.start()
t2.start()
t3.start()
t4.start()
t5.start()
t6.start()
t7.start()
```

Notice that all the above threads execute in parallel and there is not well defined sequence of temperature changes and we do not expect that the frontend gateway would receive the events in the same order as specified above in the test file. Although, we do notice that the real time clock synchronization is able to order some events (the last 4) further we notice the final temperature reported at the frontend gateway matches that specified in the above thread execution. Note: This extra test case was performed to check the limits of real clock synchronization using Berkeley algorithm.

```
Z:\UMass\DOS\code\2ndlab\spring17-lab2-hsgodhia>python ./src/gatewayfront.py localhost:9090
gateway running with below uri:
Id 2
PYRO:gateway@localhost:61834
Clock mode set to REAL
ts-logical: 1 ts-real: 18 DELIVERED (id:3,MsgType.STATE,data: 29F, lts: 3, realts: 67, from:4)
ts-logical: 2 ts-real: 19 DELIVERED (id:1,MsgType.STATE,data: 60F, lts: 1, realts: 65, from:4)
ts-logical: 3 ts-real: 20 DELIVERED (id:2,MsgType.STATE,data: 40F, lts: 2, realts: 66, from:4)
ts-logical: 4 ts-real: 21 DELIVERED (id:4,MsgType.STATE,data: 34F, lts: 4, realts: 68, from:4)
ts-logical: 5 ts-real: 22 DELIVERED (id:5,MsgType.STATE,data: 61F, lts: 5, realts: 69, from:4)
ts-logical: 6 ts-real: 23 DELIVERED (id:6,MsgType.STATE,data: 45F, lts: 6, realts: 70, from:4)
ts-logical: 7 ts-real: 24 DELIVERED (id:7,MsgType.STATE,data: 41F, lts: 7, realts: 71, from:4)
```

b) case “user enters the house”

Note: since the real clock of each process ticks by a different value (the clock drift is continuously present) between the clocks but since we continuously run Berkeley algorithm the different clock values try to converge to the average value (since Berkeley algorithm) works by taking average of the clock times and instructing each machine to set its clock to the average time. The frontend gateway receives the DELIVERED message/event in the consistent expected order of the SEND events. This confirms the working of the Berkeley algorithm based clock synchronization and successfully it orders the event to toggle the security state of the house to HOME/DISABLED when the user enters the house.

Frontend gateway snapshot

```
Clock mode set to REAL
ts-logical: 44 ts-real: 159 DELIVERED (id:1,MsgType.STATE,data: OPEN, lts: 52, realts: 159, from:3)
ts-logical: 45 ts-real: 161 DELIVERED (id:2,MsgType.STATE,data: KEY, lts: 54, realts: 161, from:3)
ts-logical: 46 ts-real: 162 DELIVERED (id:1,MsgType.STATE,data: YES, lts: 44, realts: 164, from:4)
Gateway Remotely switching ON bulb
ts-logical: 47 ts-real: 164 DELIVERED (id:3,MsgType.STATE,data: CLOSED, lts: 56, realts: 164, from:3)
ts-logical: 48 ts-real: 166 DELIVERED (id:2,MsgType.STATE,data: NO, lts: 45, realts: 167, from:4)
Gateway Remotely switching OFF bulb
```

Door sensor snapshot

```
Clock mode set to REAL
real ts 159 SEND event (id:1,MsgType.STATE,data: OPEN, lts: 52, realts: 159, from:3)
real ts 161 SEND event (id:2,MsgType.STATE,data: KEY, lts: 54, realts: 161, from:3)
real ts 164 SEND event (id:3,MsgType.STATE,data: CLOSED, lts: 56, realts: 164, from:3)
```

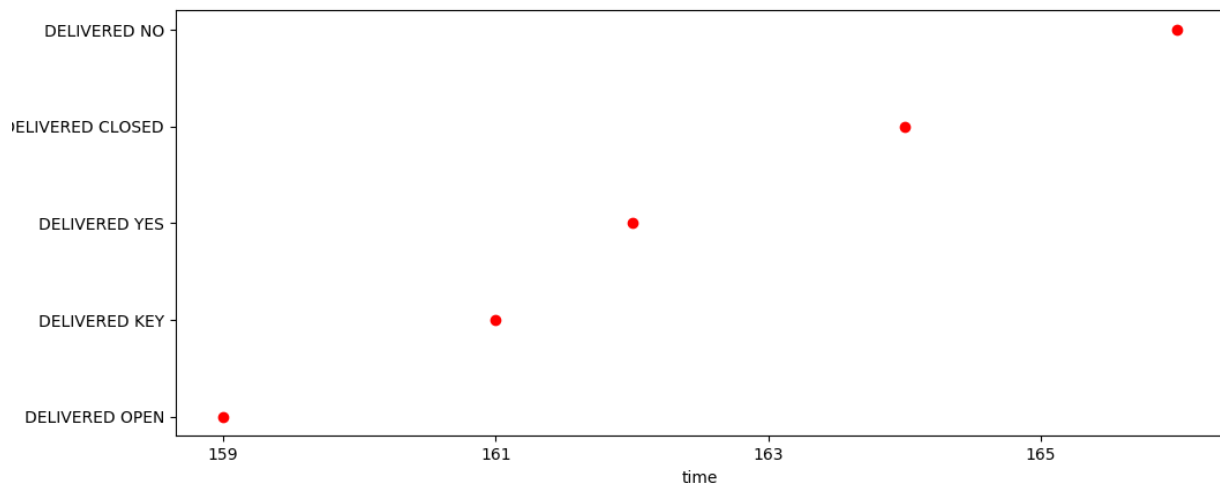
Motion sensor snapshot

```
Clock mode set to REAL
real ts: 164 SEND event (id:1,MsgType.STATE,data: YES, lts: 44, realts: 164, from:4)
real ts: 167 SEND event (id:2,MsgType.STATE,data: NO, lts: 45, realts: 167, from:4)
```

DB

```
Motion:NO Door:OPEN Temperature:29F Bulb:NA Outlet:NA logical_clock:44 real_clock:159
Motion:NO Door:KEY Temperature:29F Bulb:NA Outlet:NA logical_clock:45 real_clock:161
Motion:YES Door:KEY Temperature:29F Bulb:NA Outlet:NA logical_clock:46 real_clock:162
Motion:YES Door:CLOSED Temperature:29F Bulb:NA Outlet:NA logical_clock:47 real_clock:164
Motion:NO Door:CLOSED Temperature:29F Bulb:NA Outlet:NA logical_clock:48 real_clock:166
MSG: User Enters House, MODE: HOME, SECURTIY: DISABLED
```

Sequence of events plotted



Performance (benchmark)

We observed this test case to complete in 45 seconds

```
Z:\UMass\DOS\code\2ndlab\spring17-lab2-hsgodhia>python ./test/user_enter_exit.py
Preparing and initializing...
Starting simulation at 1490979105.1806355
End simulation at 1490979150.35349
Duration: 45.17285466194153
```

c) case “bulb lights on”

Here we test the automatic bulb switch on/off based on the messages sent by the Bulb device. We notice as expected that the DELIVERED events are in order of the SEND, indicating the working of ordering of events by the real time clock

Frontend gateway log

```
Z:\UMass\DOS\code\2ndlab\spring17-lab2-hsgodhia>python ./src/gatewayfront.py localhost:9090
gateway running with below uri:
init real clock value: 18
Id 2
PYRO:gateway@localhost:61477
Clock mode set to REAL
ts-logical: 1 ts-real: 32 DELIVERED (id:1,MsgType.STATE,data: NO, lts: 1, realts: 33, from:4)
Gateway Remotely switching OFF bulb
ts-logical: 2 ts-real: 34 DELIVERED (id:2,MsgType.STATE,data: YES, lts: 2, realts: 35, from:4)
Gateway Remotely switching ON bulb
ts-logical: 3 ts-real: 36 DELIVERED (id:3,MsgType.STATE,data: NO, lts: 3, realts: 37, from:4)
Gateway Remotely switching OFF bulb
```

Motion sensor log

```
Z:\UMass\DOS\code\2ndlab\spring17-lab2-hsgodhia>python ./src/motion_sensor.py localhost:9090
Id 4
init real clock value: 32
Sensor: Sensor running with below uri:
PYRO:Sensor.Motion@localhost:61490
Clock mode set to REAL
real ts: 33 SEND event (id:1,MsgType.STATE,data: NO, lts: 1, realts: 33, from:4)
real ts: 35 SEND event (id:2,MsgType.STATE,data: YES, lts: 2, realts: 35, from:4)
real ts: 37 SEND event (id:3,MsgType.STATE,data: NO, lts: 3, realts: 37, from:4)
```

As the frontend gateway gets the motion sensor log, it understands that it is time to remotely switch on the bulb device so remotely it starts the bulb and similarly switches it off

Bulb device log

```
Z:\UMass\DOS\code\2ndlab\spring17-lab2-hsgodhia>python ./src/bulb_device.py localhost:9090
init real clock value: 31
Id 5
Device: Bulb assigned id: 5 running with below uri.
PYRO:Device.Bulb@localhost:61495
Clock mode set to REAL
Bulb is switched: OFF
Bulb is switched: ON
Bulb is switched: OFF
```

Performance (benchmark)

We observed the lights on/off test to complete in 30seconds

```
Z:\UMass\DOS\code\2ndlab\spring17-lab2-hsgodhia>python ./test/lights_on.py
Preparing and initializing...
Starting simulation at 1490978900.1808577
End simulation at 1490978930.3437316
Duration 30.16287398338318
```

General Caveats

- We optimized the name server lookups which results in even faster performance by few seconds
- Note since we use queues, incrementing counters (variables) and a multithreaded server there is a possibility of race conditions. We avoid this and ensure it doesn't happen using global locks from the "threading" package and thread-safe synchronized queues from the "queue" package

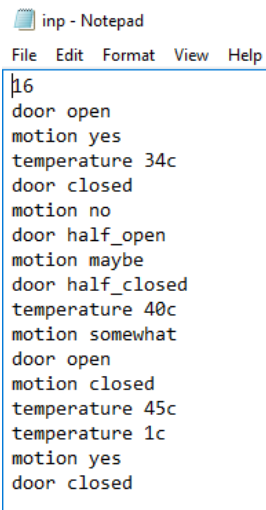
```
_msgseq_lock = threading.Lock()
_logclock_lock = threading.Lock()
_msgack_lock = threading.Lock()
```

```
def getNextLogts(self):
    global _logclock_lock
    with _logclock_lock:
        #the clock tick rate is different for each process (currently set the tick rate a
        self.logicalclock += ((self.id % 2) + 1)
        ts = self.logicalclock
    return ts
```

Gamification extra test case

Here we simulate a series of user activities in the user process like door open, closed, half-open, half-closed and other hypothetical states and scenarios. Note: this would require a sophisticated gateway since the events are hypothetical and information provided is gameified/hypothetical. This can be run from the program `python3 ./test/talk_test.py localhost:9090 < ./test/inp.txt`

The input file contains a series of commands



```
inp - Notepad
File Edit Format View Help
16
door open
motion yes
temperature 34c
door closed
motion no
door half_open
motion maybe
door half_closed
temperature 40c
motion somewhat
door open
motion closed
temperature 45c
temperature 1c
motion yes
door closed
```

Key test we observe is that all devices, sensors and gateway receive the DELIVERED events in the same consistent order. We provide the logs of a few components

Frontend gateway

```
ts-logical: 23 ts-real: 15 DELIVERED (id:1,MsgType.STATE,data: yes, lts: 1, realts: 25, from:4)
ts-logical: 24 ts-real: 18 DELIVERED (id:1,MsgType.STATE,data: open, lts: 2, realts: 21, from:3)
logical ts 25 RCVD (id:10,MsgType.STATE,data: somewhat, lts: 10, realts: 40, from:4)
ts-logical: 26 ts-real: 21 DELIVERED (id:2,MsgType.STATE,data: no, lts: 2, realts: 30, from:4)
ts-logical: 27 ts-real: 24 DELIVERED (id:1,MsgType.STATE,data: 34c, lts: 2, realts: 39, from:7)
logical ts 28 RCVD (id:12,MsgType.STATE,data: closed, lts: 12, realts: 45, from:4)
ts-logical: 29 ts-real: 27 DELIVERED (id:2,MsgType.STATE,data: closed, lts: 4, realts: 25, from:3)
logical ts 30 RCVD (id:15,MsgType.STATE,data: yes, lts: 22, realts: 55, from:4)
ts-logical: 31 ts-real: 30 DELIVERED (id:4,MsgType.STATE,data: maybe, lts: 5, realts: 35, from:4)
ts-logical: 32 ts-real: 33 DELIVERED (id:4,MsgType.STATE,data: half_open, lts: 7, realts: 29, from:3)
ts-logical: 33 ts-real: 36 DELIVERED (id:6,MsgType.STATE,data: 40c, lts: 8, realts: 47, from:7)
logical ts 34 RCVD (id:17,MsgType.STATE,data: closed, lts: 31, realts: 57, from:3)
ts-logical: 35 ts-real: 39 DELIVERED (id:10,MsgType.STATE,data: somewhat, lts: 10, realts: 40, from:4)
ts-logical: 36 ts-real: 42 DELIVERED (id:8,MsgType.STATE,data: half_closed, lts: 12, realts: 33, from:3)
ts-logical: 37 ts-real: 45 DELIVERED (id:12,MsgType.STATE,data: closed, lts: 12, realts: 45, from:4)
ts-logical: 38 ts-real: 48 DELIVERED (id:10,MsgType.STATE,data: 45c, lts: 13, realts: 55, from:7)
ts-logical: 39 ts-real: 51 DELIVERED (id:11,MsgType.STATE,data: 1c, lts: 16, realts: 63, from:7)
ts-logical: 40 ts-real: 54 DELIVERED (id:13,MsgType.STATE,data: open, lts: 19, realts: 37, from:3)
ts-logical: 41 ts-real: 57 DELIVERED (id:15,MsgType.STATE,data: yes, lts: 22, realts: 55, from:4)
ts-logical: 42 ts-real: 60 DELIVERED (id:17,MsgType.STATE,data: closed, lts: 31, realts: 57, from:3)
```

Temperature sensor

```
ts-logical: 22 ts-real: 71 DELIVERED (id:1,MsgType.STATE,data: yes, lts: 1, realts: 25, from:4)
ts-logical: 24 ts-real: 79 DELIVERED (id:1,MsgType.STATE,data: open, lts: 2, realts: 21, from:3)
logical ts 25 RCVD (id:10,MsgType.STATE,data: 45c, lts: 13, realts: 55, from:7)
logical ts 26 RCVD (id:11,MsgType.STATE,data: 1c, lts: 16, realts: 63, from:7)
ts-logical: 28 ts-real: 87 DELIVERED (id:2,MsgType.STATE,data: no, lts: 2, realts: 30, from:4)
logical ts 29 RCVD (id:10,MsgType.STATE,data: somewhat, lts: 10, realts: 40, from:4)
ts-logical: 31 ts-real: 95 DELIVERED (id:1,MsgType.STATE,data: 34c, lts: 2, realts: 39, from:7)
ts-logical: 33 ts-real: 103 DELIVERED (id:2,MsgType.STATE,data: closed, lts: 4, realts: 25, from:3)
logical ts 34 RCVD (id:12,MsgType.STATE,data: closed, lts: 12, realts: 45, from:4)
ts-logical: 36 ts-real: 111 DELIVERED (id:4,MsgType.STATE,data: maybe, lts: 5, realts: 35, from:4)
ts-logical: 38 ts-real: 119 DELIVERED (id:4,MsgType.STATE,data: half_open, lts: 7, realts: 29, from:3)
logical ts 39 RCVD (id:15,MsgType.STATE,data: yes, lts: 22, realts: 55, from:4)
ts-logical: 41 ts-real: 127 DELIVERED (id:6,MsgType.STATE,data: 40c, lts: 8, realts: 47, from:7)
logical ts 42 RCVD (id:17,MsgType.STATE,data: closed, lts: 31, realts: 57, from:3)
ts-logical: 44 ts-real: 135 DELIVERED (id:10,MsgType.STATE,data: somewhat, lts: 10, realts: 40, from:4)
ts-logical: 46 ts-real: 143 DELIVERED (id:8,MsgType.STATE,data: half_closed, lts: 12, realts: 33, from:3)
ts-logical: 48 ts-real: 151 DELIVERED (id:12,MsgType.STATE,data: closed, lts: 12, realts: 45, from:4)
ts-logical: 50 ts-real: 159 DELIVERED (id:10,MsgType.STATE,data: 45c, lts: 13, realts: 55, from:7)
ts-logical: 52 ts-real: 167 DELIVERED (id:11,MsgType.STATE,data: 1c, lts: 16, realts: 63, from:7)
ts-logical: 54 ts-real: 175 DELIVERED (id:13,MsgType.STATE,data: open, lts: 19, realts: 37, from:3)
ts-logical: 56 ts-real: 183 DELIVERED (id:15,MsgType.STATE,data: yes, lts: 22, realts: 55, from:4)
ts-logical: 58 ts-real: 191 DELIVERED (id:17,MsgType.STATE,data: closed, lts: 31, realts: 57, from:3)
```

Outlet device

```
ts-logical: 21 ts-real: 34 DELIVERED (id:1,MsgType.STATE,data: yes, lts: 1, realts: 25, from:4)
ts-logical: 22 ts-real: 41 DELIVERED (id:1,MsgType.STATE,data: open, lts: 2, realts: 21, from:3)
logical ts 23 RCVD (id:10,MsgType.STATE,data: 45c, lts: 13, realts: 55, from:7)
ts-logical: 24 ts-real: 48 DELIVERED (id:2,MsgType.STATE,data: no, lts: 2, realts: 30, from:4)
logical ts 25 RCVD (id:10,MsgType.STATE,data: somewhat, lts: 10, realts: 40, from:4)
ts-logical: 26 ts-real: 55 DELIVERED (id:1,MsgType.STATE,data: 34c, lts: 2, realts: 39, from:7)
ts-logical: 27 ts-real: 62 DELIVERED (id:2,MsgType.STATE,data: closed, lts: 4, realts: 25, from:3)
logical ts 28 RCVD (id:11,MsgType.STATE,data: 1c, lts: 16, realts: 63, from:7)
ts-logical: 29 ts-real: 69 DELIVERED (id:4,MsgType.STATE,data: maybe, lts: 5, realts: 35, from:4)
ts-logical: 30 ts-real: 76 DELIVERED (id:4,MsgType.STATE,data: half_open, lts: 7, realts: 29, from:3)
logical ts 31 RCVD (id:12,MsgType.STATE,data: closed, lts: 12, realts: 45, from:4)
logical ts 32 RCVD (id:17,MsgType.STATE,data: closed, lts: 31, realts: 57, from:3)
logical ts 33 RCVD (id:15,MsgType.STATE,data: yes, lts: 22, realts: 55, from:4)
ts-logical: 34 ts-real: 83 DELIVERED (id:6,MsgType.STATE,data: 40c, lts: 8, realts: 47, from:7)
ts-logical: 35 ts-real: 90 DELIVERED (id:10,MsgType.STATE,data: somewhat, lts: 10, realts: 40, from:4)
ts-logical: 36 ts-real: 97 DELIVERED (id:8,MsgType.STATE,data: half_closed, lts: 12, realts: 33, from:3)
ts-logical: 37 ts-real: 104 DELIVERED (id:12,MsgType.STATE,data: closed, lts: 12, realts: 45, from:4)
ts-logical: 38 ts-real: 111 DELIVERED (id:10,MsgType.STATE,data: 45c, lts: 13, realts: 55, from:7)
ts-logical: 39 ts-real: 118 DELIVERED (id:11,MsgType.STATE,data: 1c, lts: 16, realts: 63, from:7)
ts-logical: 40 ts-real: 125 DELIVERED (id:13,MsgType.STATE,data: open, lts: 19, realts: 37, from:3)
ts-logical: 41 ts-real: 132 DELIVERED (id:15,MsgType.STATE,data: yes, lts: 22, realts: 55, from:4)
ts-logical: 42 ts-real: 139 DELIVERED (id:17,MsgType.STATE,data: closed, lts: 31, realts: 57, from:3)
```

Team—Harshal Godhia, Hao Liu