Internet of Things - Smart Home Edition

Noman Bashir, Shubham Mukherjee

April 2, 2017

The goal of the project is to develop a distributed system using three sensors, two smart devices, and a multi-tier gateway. The system is intended to provide the underlying network and a set of functionalities to enable applications such as security system and occupancy-based light control. We have implemented this project as a distributed client-server system where gateway is the main server capable of communicating with the sensors and devices. We next describe the design decisions, its working, and different trade-off considered.

1 Design Decisions

We implemented this project using the Python Pyro library that enables us to implement RPCs. We implement all of our devices, sensors, and gateways as separate processes. First of all we divide the gateway to tiers: gateway processing tier (tier 1) and gateway database tier (tier 2). We assume that Pyro nameserver runs on the gateway tier 1 and all the other processes, including both gateway tiers, register themselves with it at the start. We use Bully algorithm, based on node ID, for leader election and assign the highest ID to the gateway tier 1 so that it is always selected as the leader. We argue that it is logical to assume that gateway tier 1 will almost always have the highest .

Apart from same implementation of leader election, clock sync, logical clocks, and registering with gateway, our implementation provides a few interfaces that are same for all the sensors, devices, and gateway processes. These interfaces are described below:

- **receiveMessage(message):** this interface is responsible for handling the leader election, clock synchronization, and logical clock messages. This function also handles the push and response messages sent by the sensors and devices to the gateway. Upon receiving the message, this interface calls respective functions to take appropriate action.
- **sendMessage(msg):** this interface is responsible for sending all the messages related to leader election, clock sync, and logical clocks. This function provides a mechanism to send a message to a particular process, a set of processes, or simply broadcast. This method also handles the push messages by the sensors who intend to send their value to the gateway.
- **getState():** this interface provides gateway a mechanism to get the state of the sensor or a device. It reports the state when gateway calls this function to query the state.

1.1 Sensors

We implement four sensors in our project: door sensor, motion sensor, temperature sensor, and presence sensor. They all share the above mentioned interfaces. The sensor specific interfaces are described below:

1.1.1 Motion Sensor

Motion sensor provides an interface, triggerMotion(), that is used by the user process to trigger the motion sensor. After it senses the motion, it sends a message to the gateway.

1.1.2 Door Sensor

It provides an interface, openDoor(ID), which is called by the user to open the door. We assume that user provides its ID through the beacon attached to the key, if it is verified as a valid ID then the door is opened and gateway is informed. However if the ID is not confirmed, the security system starts ringing the alarms and intruder is denied the access.

1.1.3 Temperature Sensor

This sensor implements a senseTemperature() interface that currently generates a random number in a specified range. This interface is used by the getState() function to get temperature value when gateway asks for it. In its current version, it doesn't report its state to the gateway automatically.

1.1.4 Presence Sensor

It provides an interface named, detect_presence(), to the user. Whenever a valid user wants to enter the home, his/her presence is sensed by the presence sensor. It sends the update to the gateway to inform that valid user will try to open the door now. We assume that this presence sensor is triggered before the user attempts to open the door.

1.2 Smart Devices

There are two smart devices implemented in this project: smart bulb and smart outlet. These devices also provide the receiveMessage(message), sendMessage(message), and getState() interfaces just like sensors. Unlike sensors, both devices offer a same interface for their core functionality:

- **controlState()** this interface can be used by both the gateway and the user to change the state of the device. Whenever the device's state is changed, it sends a message to the gateway about the state change.

1.3 Gateway

We have divided the gateway to two separate tiers capable of running on different machines. We next describe the gateway specific functionalities offered by these tiers.

1.3.1 Gateway Processing Tier

The main functionalities of the gateway tier 1 include communicating with sensors and devices, storing and retrieving the data from database, and implementing the core logic of the security and home automation system. In order to enable these tasks, gateway tier 1 implements the following interface:

- **queryState(device ID):** this function is called by the core logic of the program to get the state of a particular device are sensor as needed. It calls the getState() function which responds by using the sendMessage(message) mechanism.
- **process_state_response(message):** all of the response messages received by the receiveMessage(message) interface are forwarded to this interface. which simply prints the information as of now.
- **sendControlMsg(message):** this interface is used by the gateway to control the state of smart devices. It ensures that the change of state message is forwarded only to the smart devices.
- **store(message):** this interface is responsible for saving all the push messages and logs at the database tier of the gateway. It calls the insert function at tier 2 to save the information on the file.
- **retrieve():** this interface enables the retrieval of last message from the database tier. However, in future, it can be modified to retrieve the desired information based on the timestamp, device ID, and other parameters.
- **eventOrderLogic(message):** this function implements the core logic of the project which enables a security and home automation system. This function is called whenever the door or motion sensor send s push message to update their state. It checks the valid ID, state of the home, and infers whether the user is leaving the home or entering. It furthers controls security system and state of the smart bulb.

1.3.2 Gateway Database Tier

The tier two of the gateway is responsible for storing the events and enabling the retrieval of this data. In order to implement this functionality, tier 2 implements the following functions.

- **insert(message):** this message is called by the store(message) function at tier 1 and inserts every event on a new line in the database file.
- **retrieve():** this interface enables the gateway tier 1 to retrieve the last event information from the database.
- **retrieve_all():** this interface enable the gateway tier 1 to retrieve all of the data stored in the database file.

1.3.3 Leader Election and Clock Synchronization

For the leader election algorithm, we use Bully algorithm based on process ID. The following interfaces are implemented as a part of the leader election algorithm:

- **init_election():** this function sends the leader election message to all of the other processes running in the id list. Also, if it doesn't receive reply from anyone or receives an election done message, it sets itself as a leader and propagates a leader win message.
- **process_election():** this function is called if a leader election message is received at any of the processes. It sends acknowledgment to the sender and starts an election of its own.
- **set_leader():** this function is called when a leader win message is received at any of the processes. It will designate the sender as the leader in the system.
- **ok():** this method is called by the process election function to send a reply to the election initiator. If the election is done it simply returns, otherwise, it sets the recv_ok as true.

Once the leader is elected, it starts the clock synchronization process which is enabled by the functions described below:

- **poll_clocks():** this function implements the core logic of clock synchronization. Every 5 seconds leader asks the other clocks to send their clocks. It then runs the standard Berkeley clock synchronization algorithm to compute the offset, which is the sent to all the other processes.
- **send_timestamp():** this function runs at the non-leaders processes and responds to the poll clocks message by sending the current timestamp at the process.

1.3.4 Logical Clocks

In order to implement the logical clocks, we used a modified version of the totally ordered multicast algorithm with Lamport clocks. Whenever an event occurs, local event counter is updated by one and a message is sent to all the other processes. When they receive the message, they update their counter by one if it has not been already done by the process itself. This is used to ensure that counter is consistent across all processes. This mechanism is enabled by the following functions:

- **logicalClock(message):** this function is called when the main function of any of the processes is occurs i.e. door sensor is triggered. It sends a clock update message to all the other processes who update their clocks accordingly.

2 How it works?

The process to run a test case for this project is explained in the README file. However, In each of the test cases, the flow of actions can be described as follows:

1. Pyro nameserver starts at localhost or at IP address of the gateway tier 1.
2. All of the processes register themselves with the nameserver. Currently, we have a single script to register all the processes. However, it can be easily modified to register individual processes at separate machines.
2. All of the processes are assigned an ID based on the configuration file.
3. Election is started by all the nodes to select the leader for clock synchronization algorithm.
4. After the leader is elected, clock synchronization is done on periodic basis. Alternatively, event based logical clock is used.
5. Whenever a sensor or a device sends the push message, the incoming message is stored in to the database.
5. The event order logic is run to change the state of security system and smart bulb, if needed.
6. The process continues as long as the user is performing any actions.

3 Design Trade-offs and Assumptions

There are some trade-offs and assumptions that we made during the design of the project. The details of these trade-offs is explained below:

- **Knowledge of the system:** We assume that all of the processes know the IDs of other processes. Currently we read those from a configuration file, however they can be easily fetched from the nameserver or the gateway.
- **Assigning IDs:** We assigned static IDs to all the processes when they registered themselves with the nameserver. We considered using a more dynamic mechanism to assign IDs but that would have increased the complexity of the problem. We left this task for the future work.
- **Leader election:** We used Bully algorithm for leader election that used processes ID as the metrics for leader selection. We considered using a more sophisticated metrics for the task, but it was left for future work.

- **Logical clocks:** While implementing logical clocks, we used a modified version of totally ordered multicast algorithm. Whenever a event occurs at a process i, it sends a message to all the other processes to increases their clocks. We don't implement the ack mechanism as it doesn't work for the system under consideration. If an ack is lost in original mechanism, the update is not pushed to the application layer. We cannot afford to do that as the event has already occurred and we cannot reverse it even if ack fails.

- **Door sensor:** We implement the functionality of door sensor to detect door open event rather than door status. Whenever a user comes in or goes out, the door sensor senses a door opening event and door is assumed to be closed soon after.

- **Sensor placement:** We assume that presence sensor is triggered before the door sensor if the user is coming from outside. Also, we assume that motion sensor is placed in the corridor right after the door so, in the event of a user entering or leaving, both of the sensors are triggered one after the other.

- **Temperature Sensor:** Although we have implemented the temperature sensor as both pull and push based sensor, but we use it as the pull-based sensor only. We leave the periodic updates from the temperature sensor as future tasks.

4 Possible Improvements and Extensions

Our project has a strong core which can enable a variety of applications for a smart home. Ideally, the first step would be to handle the assumption that we made during this project as **outlined in the previous section**. However, some of the further possible improvements in the core system as well as application are envisioned below:

- In order to enable variety of smart home solutions, we need to implement a better data storage and retrieval mechanism. We can achieve this by using python friendly databases like SQlite or PostgreSQL.

- We can assume a variety of devices connected to the smart outlet and then build creative applications on top of it. For example, we can assumer a heater/AC connected to the outlet and control its state based on the temperature value from the sensor.

- Currently we demonstrate the performance of leader election algorithm and clock synchronization using only the on screen print messages. However, a separate test to check the integrity of these algorithms in future.