Internet of Things Apps Platform

Project Design Document
Version-1.0

For the module
Gateway

Submitted by

Team 3 :

Abhinaba Sarkar	201405616
Atul Rajmane	201405529
Prerna Chauhan	201405544
Aditi Jain	201405549

Of
Group Number - 5

Under the guidance of
Mr. Ramesh Loganathan

For the course
Internals of Application Server
IIIT, Hyderabad

23rd March, 2015

Contents

List of Figures

Chapter 1

Low Level Design

1.1 Lifecycle of the module

1.1.1 Repository Server

- First the Repository sever will be turned on. It will be configured with some of the static information such as type handlers ,and other config files.

1.1.2 Registry Server

- Registry server will be turned on. It will have the current network scenario of gateways and sensors.
- It will have a table in which corresponding to each gateway there will be a list of sensors that are connected to a particular gateway, along with sensor IDs it will also contain metadata for sensor such as its sensor type, last uptime etc. The information in registry will be updated according to the heartbeat messages sent by the gateways periodically.

- It will have a table which will store the gateway IDs and there corresponding hardware IDs, this table is used to authenticate gateways. This table will be updated whenever a new gateway joins the system or leaves the system.

1.1.3 Gateways

- Gateways will be turned on which will be responsible for collecting data from set of sensors which are connected to it and then forwarding this data to the filter sever.
- It also sends the heartbeat message (which contain information about the sensors connected to that gateway) to the Registry sever according to which registry is updated.
- It maintains a table for storing information about the sensors connected to it. Table contains fields like sensorID, type, last uptime, etc.
- Whenever a new gateway is brought to the system tables in registry server is updated according to the information about new gateway such as Hardware ID , sensors connected to that gateway etc, is added to the system. Also once it successfully registers itself to registry server , Gateway loads all the type handlers from the repository server.
- Whenever a gateway leaves the system tables in registry are updated accordingly. The status corresponding to that particular gateway is changed, also status corresponding to the sensors attached to that gate-

way is changed to Not Available.

1.1.4 Sensors

- Sensors are responsible for generating data according to the event occurred.
- Whenever a new sensor is added to the system it is configured with the gateway address to which it will send its data, also once it has gateway address it will register itself to the gateway by sending an introduction message which will have information like sensorID, type of sensor etc.
- Sensor sends heartbeat message to the gateway and according to which gateway updates its table for sensor information.

Chapter 2

Sub modules

2.1 Interactions between SubModules

2.1.1 Interaction between sensor and gateways

- Whenever new sensor comes to the system it will be configured according to the nearest gateway in the same location, and then it will register itself to the gateway registry.
- After successful registration of sensor, sensor will periodically send sensed data to the gateway, and gateway forwards this data to the filter server.
- If the request comes for the data of specific sensor then gateway will interact with that sensor in order to get the latest data from that sensor.
- Periodically sensor will send heartbeat message to the gateway , according to which gateway maintains its registry.

2.1.2 Interactions between Gateways and Registry Server

- Whenever a new gateway is installed in the system it will register itself to the registry server, which has gateway information table stored in it.
- Periodically gateways will send heartbeat messages to the registry server according to which registry server will maintain its tables. These message will contain the sensor information corresponding to that gateway, according to this message sensor information table stored in registry server is also updated.

2.1.3 Interactions between Gateways and Filter Server

- Whenever a request comes to the filter server , according to the location in request packet ,filter server will consult registry server for finding the gateway addresses corresponding to the requested location and then it will unicast the packet to the corresponding gateways.
- Gateway forwards the sensor data packets to the filter server, which stores this data to its database.

2.1.4 Interactions between two gateways

- Gateways are connected in meshed topology in our system. Routing algorithm is implemented on each of them. They route packet sent by other gateway to its destination that is either to the filter server or to the registry server.

2.1.5 Interactions between Gateways and Repository Server

- Gateways interact with the repository so as to load the type handlers, these will be used by gateways to handle different type of sensor devices such as BLE, Wi-Fi etc.

Chapter 3

Overview of each module

3.1 Sensors

3.1.1 Functionality

- The sensors when added to the system, are configured in such a way that they can send the data to their respective gateways.
- When they are up, they send the respective gateways the following information in a data packet - (sensorID, type, hardwareID)
- On receiving this information, the gateway verifies this data with the repository server.
- The sensors keep on generating data after certain time intervals as mentioned in their configuration and sends them to the respective gateway.
- The sensors send heartbeat messages to the gateway after every small interval of time
- These heartbeat messages are forwarded to the registry server which maintains the status information for the sensors.

3.2 Gateways

3.2.1 Functionality

- The main purpose of the gateways is to make the data sent by sensors 'internet ready'
- Gateways have type handlers, which convert the data from the sensors to TCP packets according the protocol used by the sensors.
- Whenever the gateway receives any data from the sensors, it forwards them to the filter server after repackaging it.
- Whenever it receives heartbeat messages from the sensors, it forwards them to the registry server for status updation.
- Whenever a new sensor is added to the system, sensor sends an introduction message to the respective gateway which in turn verifies it from the repository server.
- The gateways themselves also send heartbeat pings to the registry server after a certain interval to maintain the status information.

3.2.2 Block Diagram

3.2.3 Interactions with other modules

- **Sensors** The sensors send three type of messages to the gateway - introductory, heartbeat and data message. The gateway receives the message through appropriate type handler and repackages the data in a

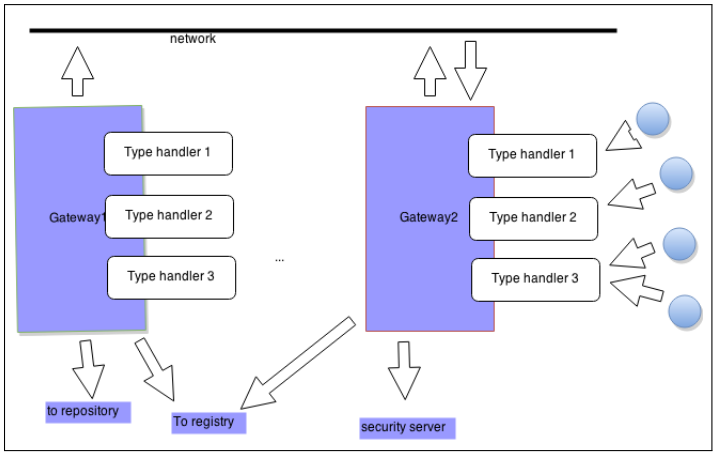

Figure 3.1: *Block Diagram of Gateways*

TCP header. The introductory message is then forwarded to repository server for verification. The heartbeat pings are sent to the registry server for status update. The data packets are sent to the filter server.

- **Repository Server** During boot up, the gateway connects to the repository server to load the type handler modules from the repository server. Also, whenever a new sensor sends a introductory message to the gateway, the gateway validates the static data with the repository server.

- **Registry Server** Whenever the gateway receives heartbeat message from the sensors, the gateway converts the data packet and sends the status to the registry server which in turn updates the status information. Also, the gateway sends heartbeat pings to the registry server which updates the status for the gateways.

- **Filter Server** The sensor data that is continuously obtained by the gateways are sent to the filter server after changing the data format. Also after every pre decided time interval, the gateway checks for any

pending command in the command queue of the filter server. On receiving any command, the gateway sends the request to the sensors and returns the data to the filter server.

- **Security Server** Gateway gets a session token from the security server, the first time it boots up. The same session token is used for any communication with the filter server.

- **Other Gateways** Gateways are connected in meshed topology in our system. Routing algorithm is implemented on each of them. They route packet sent by other gateway to its destination that is either to the filter server or to the registry server.

3.3 Repository Server

3.3.1 Functionality

- The repository server contains all the static information about the components of the system.

- For the sensors it contains the information like type of the sensor, itâĂŹs communication protocol, location, etc.

- For the gateways it contains the sensors its connected to, etc.

- It also contains the different protocol handlers for the gateways. Whenever a gateway needs to handle a sensor with a certain type of protocol, it loads the type handler from the repository server.

3.3.2 Interactions with other modules

- **Gateway** Whenever new sensors are added to the system and they send a introduction message to the gateway, the gateway validates that information from the repository server. The repository server exposes a REST API that is consumed by the gateway. Also, the gateway requests for a certain type handlers when the gateway boots up. The code for the type handlers is also sent through REST based APIs to the gateways.

- **Admin Panel** The admin may add new sensors or gateways through the sensor admin panel. This, in the background updates the repository information.

3.3.3 APIs

1. /v1/<sensor_id> "sensor_type" will be the one registered in the sensor registry.

   Eg: /v1/sensor45, etc.

   The above API will check whether the sensor is already registered with the repository.

   Response json is below :

```
{
    "version": "1.0",
    "timestamp": "<response timestamp generated at server>",
    "comments": "<comment specific to this response, can be empty>",

    "request": {
    "sensor_id" : "<sensor_id sent in the request>"
    },

"data": {
    "sensors-data" : [
    {"sensorId" : "<Id>",
    "sensor_type" : "<type of the sensor>",
    "protocol" : "<protocol for the sensor>",
    "locationId":"<location Id>"
    },
    ]
}
}
```

Listing 1: The response of /v1/<sensor_id>

2. /v1/<sensor_protocol> "sensor_protocol" will be the one registered in the sensor registry.

   Payload -

   access_key: <secret key which the app uses to communicate with API>

   Eg: /v1/handler/ble, /v1/handler/wifi

   This API will send the javascript code pertaining to the specific type of protocol requested.

```
{
    "version": "1.0",
    "timestamp": "<response timestamp generated at server>",
    "comments": "<comment specific to this response, can be empty>",

    "request": {
    "sensorProtocol": "<sensor_protocol_sent_in_request>",
    },

"data": {
    "sensors-data" : [
    {"sensorProtocol" : "<Protocol>",
    "handler" : "<javascript code for the handler>"
    }
    ]
}
}
```

Listing 2: The response of /v1/<sensor_protocol>

3. /v1/repository/<sensor_typeid> "sensor_typeid" will be the one registered in the sensor registry.

   Payload -

   access_key: <secret key which the gateway uses to communicate with API>

   Eg: /v1/repository/2, etc.

   This API returns all the information about a certain sensor type. This API forms a part of the larger **QUERY API**

```
{
    "version": "1.0",
    "timestamp": "<response timestamp generated at server>",
    "comments": "<comment specific to this response, can be empty>",
    "request": {
    "typeId": "<sensor_typeid_sent_in_request>",
},
"data": {
    "sensors-data" : [
    {"sensorId" : "<Id>",
    "sensor_type" : "<Type of the sensor>",
    "sensor_protocol" : "<protocol used by the sensor>",
    "recording" : "<composite object specific to type of sensor>",
    "timestamp" : "<timestamp of recording by sensor>"
    },
    {"sensorId" : "<Id>",
    "sensor_type" : "<Type of the sensor>",
    "sensor_protocol" : "<protocol used by the sensor>",
    "recording" : "<composite object specific to type of sensor>",
    "timestamp" : "<timestamp of recording by sensor>"
    },
    ...,
    ...,

    ]
}
}
```

Listing 3: The response of /v1/repository/<sensor_typeid

4. /v1/repository/gateway

Payload -

access_key: <secret key which the gateway uses to communicate with API>

This API returns all the information the registered gateways. This API forms a part of the larger **QUERY API** to be called by the filter server.

```
{
    "version": "1.0",
    "timestamp": "<response timestamp generated at server>",
    "comments": "<comment specific to this response, can be empty>",
"data": {
    "gateway-data" : [
    {"gatewayId" : "<Id>",
    "registered_sensorids" : ["<Array of sensor ids>"],
    },
    {"gatewayId" : "<Id>",
    "registered_sensorids" : ["<Array of sensor ids>"],
    },
    ...,
    ...,

    ]
}
}
```

Listing 4: The response of /v1/repository/gateway

3.4 Registry Server

3.4.1 Functionality

- The registry server contains the current status information of the gateways and the sensors

- The continuous health pings received by the sensors are forwarded to the registry server. The registry server updates the uptime and last received ping time in the server.
- The gateway also sends health pings to the registry which is also updated.
- The registry is also updated through the admin panel when the admin adds a new sensor or a gateway

3.4.2 Interactions with other modules

- **Gateway** The gateway sends heartbeat messages for itself and also forwards the heartbeat messages sent by the sensors.
- **Admin Panel** The platform admin can add or remove sensors and also add or remove gateways as required
- **Filter Server** The filter server asks the registry server for the status of a particular sensor or a gateway. This interaction takes place through a REST API.

3.4.3 APIs

1. /v1/registry/sensor_status/<sensor_id> "sensor_id" will be the one registered in the registry.

   Payload -

access_key: <secret key which the gateway uses to communicate with API>

Eg: /v1/registry/sensor_status/sensor4, etc.

This API returns last known information about a certain sensor

```
{
    "version": "1.0",
    "timestamp": "<response timestamp generated at server>",
    "comments": "<comment specific to this response, can be empty>",
    "request": {
    "Id": "<sensor_id_sent_in_request>",
},
"data": {
    "sensors-data" : [
    {"sensorId" : "<Id>",
    "sensor_type" : "<Type of the sensor>",
    "sensor_protocol" : "<protocol used by the sensor>",
    "information" : "<last updated battery information of the sensor",
    "timestamp" : "<timestamp of recording by sensor>"
    },
    ]
}
}
```

Listing 5: The response of /v1/registry/sensor_status/<sensor_id>

2. /v1/registry/active_sensor/

Payload -

access_key: <secret key which the filter server uses to communicate with the API>

The above API will give all information about all the active sensors irrespective of types. This forms a part of **QUERY API**.

```
{
    "version": "1.0",
    "timestamp": "<response timestamp generated at server>",
    "comments": "<comment specific to this response, can be empty>",

"data": {
    "sensors-data" : [
    {"sensorId" : "<Id>",
    "sensor_type" : "<Type of the sensor>",
    "sensor_protocol" : "<protocol used by the sensor>",
    "recording" : "<composite object specific to type of sensor>",
    "timestamp" : "<timestamp of recording by sensor>"
    },
    {"sensorId" : "<Id>",
    "sensor_type" : "<Type of the sensor>",
    "sensor_protocol" : "<protocol used by the sensor>",
    "recording" : "<composite object specific to type of sensor>",
    "timestamp" : "<timestamp of recording by sensor>"
    },
    ...,
    ...,

    ]
}
}
```

Listing 6: The response of /v1/registry/active_sensor/

3. /v1/registry/sensor_typeid/

   Payload -

   access_key: <secret key which the filter server uses to communicate with the API>

   The above API will give information about all the sensors of a certain type. This forms a part of **QUERY API**.

   The response for the above is same as the response jsons captured above

4. /v1/registry/sensor_typeid/active

   Payload -

   access_key: <secret key which the filter server uses to communicate with the API>

   The above API will give information about all the sensors of a certain type which are active. This forms a part of **QUERY API**.

   The response for the above is same as the response jsons captured above

Chapter 4

Interactions between sub modules

4.1 Interactions between SubModules

4.1.1 Interaction between sensor and gateways

- Whenever new sensor comes to the system it will be configured according to the nearest gateway in the same location, and then it will register itself to the gateway registry.
- After successful registration of sensor, sensor will periodically send sensed data to the gateway, and gateway forwards this data to the filter server.
- If the request comes for the data of specific sensor then gateway will interact with that sensor in order to get the latest data from that sensor.
- Periodically sensor will send heartbeat message to the gateway , according to which gateway maintains its registry.

4.1.2 Interactions between Gateways and Registry Server

- Whenever a new gateway is installed in the system it will register itself to the registry server, which has gateway information table stored in it.
- Periodically gateways will send heartbeat messages to the registry server according to which registry server will maintain its tables. These message will contain the sensor information corresponding to that gateway, according to this message sensor information table stored in registry server is also updated.

4.1.3 Interactions between Gateways and Filter Server

- Whenever a request comes to the filter server , according to the location in request packet ,filter server will consult registry server for finding the gateway addresses corresponding to the requested location and then it will unicast the packet to the corresponding gateways.
- Gateway forwards the sensor data packets to the filter server, which stores this data to its database.

4.1.4 Interactions between two gateways

- Gateways are connected in meshed topology in our system. Routing algorithm is implemented on each of them. They route packet sent by other gateway to its destination that is either to the filter server or to the registry server.

4.1.5 Interactions between Gateways and Repository Server

- Gateways interact with the repository so as to load the type handlers, these will be used by gateways to handle different type of sensor devices such as BLE, Wi-Fi etc.

4.2 Application Programming Interface

4.2.1 Repository Server APIs

1. /v1/<sensor_id> "sensor_type" will be the one registered in the sensor registry.

   Eg: /v1/sensor45, etc.

   The above API will check whether the sensor is already registered with the repository.

   Response json is below :

```
{
    "version": "1.0",
    "timestamp": "<response timestamp generated at server>",
    "comments": "<comment specific to this response, can be empty>",

    "request": {
    "sensor_id" : "<sensor_id sent in the request>"
    },

"data": {
    "sensors-data" : [
    {"sensorId" : "<Id>",
    "sensor_type" : "<type of the sensor>",
    "protocol" : "<protocol for the sensor>",
    "locationId":"<location Id>"
    },
    ]
}
}
```

Listing 7: The response of /v1/<sensor_id>

2. /v1/<sensor_protocol> "sensor_protocol" will be the one registered in the sensor registry.

   Payload -

   access_key: <secret key which the app uses to communicate with API>

   Eg: /v1/handler/ble, /v1/handler/wifi

   This API will send the javascript code pertaining to the specific type of protocol requested.

```
{
    "version": "1.0",
    "timestamp": "<response timestamp generated at server>",
    "comments": "<comment specific to this response, can be empty>",

    "request": {
    "sensorProtocol": "<sensor_protocol_sent_in_request>",
    },

"data": {
    "sensors-data" : [
    {"sensorProtocol" : "<Protocol>",
    "handler" : "<javascript code for the handler>"
    }
    ]
}
}
```

Listing 8: The response of /v1/<sensor_protocol>

3. /v1/repository/<sensor_typeid> "sensor_typeid" will be the one registered in the sensor registry.

   Payload -

   access_key: <secret key which the gateway uses to communicate with API>

   Eg: /v1/repository/2, etc.

   This API returns all the information about a certain sensor type. This API forms a part of the larger **QUERY API**

```
{
    "version": "1.0",
    "timestamp": "<response timestamp generated at server>",
    "comments": "<comment specific to this response, can be empty>",
    "request": {
    "typeId": "<sensor_typeid_sent_in_request>",
},
"data": {
    "sensors-data" : [
    {"sensorId" : "<Id>",
    "sensor_type" : "<Type of the sensor>",
    "sensor_protocol" : "<protocol used by the sensor>",
    "recording" : "<composite object specific to type of sensor>",
    "timestamp" : "<timestamp of recording by sensor>"
    },
    {"sensorId" : "<Id>",
    "sensor_type" : "<Type of the sensor>",
    "sensor_protocol" : "<protocol used by the sensor>",
    "recording" : "<composite object specific to type of sensor>",
    "timestamp" : "<timestamp of recording by sensor>"
    },
    ...,
    ...,

    ]
}
}
```

Listing 9: The response of /v1/repository/<sensor_typeid

4. /v1/repository/gateway

   Payload -

   access_key: <secret key which the gateway uses to communicate with API>

   This API returns all the information the registered gateways. This API forms a part of the larger **QUERY API** to be called by the filter server.

```
{
    "version": "1.0",
    "timestamp": "<response timestamp generated at server>",
    "comments": "<comment specific to this response, can be empty>",
"data": {
    "gateway-data" : [
    {"gatewayId" : "<Id>",
    "registered_sensorids" : ["<Array of sensor ids>"],
    },
    {"gatewayId" : "<Id>",
    "registered_sensorids" : ["<Array of sensor ids>"],
    },
    ...,
    ...,

    ]
}
}
```

Listing 10: The response of /v1/repository/gateway

4.2.2 Registry Server APIs

1. /v1/registry/sensor_status/<sensor_id> "sensor_id" will be the one registered in the registry.

   Payload -

access_key: <secret key which the gateway uses to communicate with API>

Eg: /v1/registry/sensor_status/sensor4, etc.

This API returns last known information about a certain sensor

```
{
    "version": "1.0",
    "timestamp": "<response timestamp generated at server>",
    "comments": "<comment specific to this response, can be empty>",
    "request": {
    "Id": "<sensor_id_sent_in_request>",
},
"data": {
    "sensors-data" : [
    {"sensorId" : "<Id>",
    "sensor_type" : "<Type of the sensor>",
    "sensor_protocol" : "<protocol used by the sensor>",
    "information" : "<last updated battery information of the sensor",
    "timestamp" : "<timestamp of recording by sensor>"
    },
    ]
}
}
```

Listing 11: The response of /v1/registry/sensor_status/<sensor_id>

2. /v1/registry/active_sensor/

   Payload -

   access_key: <secret key which the filter server uses to communicate with the API>

   The above API will give all information about all the active sensors irrespective of types. This forms a part of **QUERY API**.

```
{
    "version": "1.0",
    "timestamp": "<response timestamp generated at server>",
    "comments": "<comment specific to this response, can be empty>",

"data": {
    "sensors-data" : [
    {"sensorId" : "<Id>",
    "sensor_type" : "<Type of the sensor>",
    "sensor_protocol" : "<protocol used by the sensor>",
    "recording" : "<composite object specific to type of sensor>",
    "timestamp" : "<timestamp of recording by sensor>"
    },
    {"sensorId" : "<Id>",
    "sensor_type" : "<Type of the sensor>",
    "sensor_protocol" : "<protocol used by the sensor>",
    "recording" : "<composite object specific to type of sensor>",
    "timestamp" : "<timestamp of recording by sensor>"
    },
    ...,
    ...,

    ]
}
}
```

Listing 12: The response of /v1/registry/active_sensor/

3. /v1/registry/sensor_typeid/

   Payload -

   access_key: <secret key which the filter server uses to communicate with the API>

   The above API will give information about all the sensors of a certain type. This forms a part of **QUERY API**.

   The response for the above is same as the response jsons captured above

4. /v1/registry/sensor_typeid/active

   Payload -

   access_key: <secret key which the filter server uses to communicate with the API>

   The above API will give information about all the sensors of a certain type which are active. This forms a part of **QUERY API**.

   The response for the above is same as the response jsons captured above