A close up of a logo

Description automatically generated

**Mobile Computing WS-19/20– IoT Smart Home**

**Fire Alarm And Prevention**

**Name : Devan Premchandrakumar Nair**

**Matriculation Number : 1274369**

**Guidance : Mr. Gregor Frick**

**Mr. Besfort Shala**

**Prof. Armin Lehmann**

**Table of Contents**

[Case Description 3](#_Toc29570100)

[Network Architecture 4](#_Toc29570101)

[**Sensor-Actuator Emulator** 4](#_Toc29570102)

[**IoT Gateway** 4](#_Toc29570103)

[Protocol Descriptions 5](#_Toc29570104)

[**CoAP** 5](#_Toc29570105)

[**HTTP 1.1** 7](#_Toc29570106)

[Implementation Strategy 8](#_Toc29570107)

[**IoT Gateway Architecture** 8](#_Toc29570108)

[**Resource Discovery** 8](#_Toc29570109)

[**State Generation** 9](#_Toc29570110)

[**Sensor Actuator Emulator Environment** 12](#_Toc29570111)

[Project Setup Instructions 16](#_Toc29570112)

[**Ubuntu** 16](#_Toc29570113)

[**Windows** 16](#_Toc29570114)

[Implementation Analysis 17](#_Toc29570115)

[***Default case(version-1)*** 17](#_Toc29570116)

[***version-2(Non-default case)*** 23](#_Toc29570117)

[CoAP URI Summary 25](#_Toc29570118)

[HTTP URI Summary 26](#_Toc29570119)

[Appendix A 27](#_Toc29570120)

[Acknowledgements 27](#_Toc29570121)

[References 28](#_Toc29570122)

# **Case Description**

For the given case of smart fire alarm – prevention and detection, we consider a scenario of a home with multiple rooms. In our case, for the simulations we have chosen three rooms. The rooms are labelled as ‘Room 1’,’Room 2’and ‘Room 3’. Each of these rooms are equipped with the same nature of sensors and actuators. The sensors in these rooms are designed to emulate, measurement of a set of environment parameters namely – temperature (in deg Celsius),flash value (IR-spectrum intensity sensors) , smoke sensor (a finite state machine with 2 states – ‘minimal’ and ‘intense’). The actuators in the room are used to emulate a fire alarm(notification about each room), a smart-extinguisher actuator (which activates extinguisher for a specific volume rate and specific time). The activation of extinguisher actuator is expected to reduce the effects fire has on sensor parameters. So the sensors should start reading reduced values once the actuator starts working.

This use case proposes to emulate two scenarios – one where there is fire in one of the rooms(‘Room 3’ which is the default or *version-1* case ) or another where there is fire in two rooms(‘Room 2’ and ‘Room 3’ also referred in this document as *version-2* or non-default case) . The gateway detects a fire through its measurement value retrieved from the sensors. The gateway then triggers the appropriate alarm related to the room which has the fire, thereby enabling client to know which room the fire has taken place and will also trigger an actuator that will extinguish the fire. The gateway will prompt the user for confirmation for actuation in the room once it recognizes that sensor measurement values for environment parameters exceeds the threshold values. This is to simulate a case of user-confirmation to make sure if there was an actual fire and it is not because of malfunctioning(hypothetical) of one of the sensors. After the user has prompted affirmation to extinguish the fire one can observe the temperature reducing in the respective room and the smoke reducing thereby emulating actual extinguishing of a fire before it progresses to a catastrophic end. It also enables a set of guidance actuators which will guide possibly trapped occupants of the room to emergency exits or some other mechanism which achieves a similar goal. It will trigger an alarm indicating the location of the room with the fire. The user may also deny permission for fire extinguisher which will prevent extinguisher from operation ,if the client-user confirmed in person that there was no fire and that excess measurement reading of environment parameters were a result of faulty sensors – a highly likely scenario since these sensors are battery-operated autonomous Internet connected devices . In the non-default case irrespective of the fact whether the user gave permission to raise an alarm(or extinguish fire) the gateway will prioritize over the fact that it is highly unlikely that sensors across multiple rooms failed and provided a false positive and the most meaningful interpretation could be that there is a massive fire and so it triggers actuation and alarms in the two rooms and try to extinguish the fire irrespective of the option the user responded to the prompt with. The IoT-Gateway has a web console which can be opened in an Internet browser to view the sensor measurements and actuator states in real-time and to respond to gateway alerts and prompts.

This document is expected to serve as a manual for operation and provides architectural description of the project. The description for the functionality and attributes are expressed in a programming language abstract fashion.

# **Network Architecture**

HTTP

Web Browser

IoT Gateway

Sensor-Actuator Emulator

CoAP

*{xml-payload}*

*{json-payload}*

localhost:8080

localhost:62235

localhost:5685,..,5699

Figure : Network Diagram

The project has two major components – Emulator for sensor-actuator environment and IoT gateway.

## **Sensor-Actuator Emulator**

The emulator has all the sensors and actuators related to the use-case residing in localhost. The different sensors and actuators have distinct L4 addresses (the port number range [5685-to-5699] and description for those are given in [Appendix A](#_Appendix_A)). The emulator nodes all use *coap* over UDP scheme. Each sensor and actuator runs a CoAP resource on a CoAP server(Datagram Server) on a single port performing its respective function(reading temperature, luminosity etc. ). The payload that is sent or received by the hosts in the emulator is of JSON content format.

## **IoT Gateway**

The IoT gateway has two interfaces . As shown in the figure the interface communicating via CoAP over UDP, resides on localhost:62235. It communicates to the sensors and actuators residing in the emulator environment. It hosts multiple CoAP client services and subscription listeners on a single endpoint at localhost:62235.It communicates with the emulator environment with JSON payload content format. The gateway also has a web-console which it hosts at the address localhost:8080. The web-console resides under the URL- *http://localhost:8080/console*. The gateway communicates with the browser over this interface via HTTP/1.1 ReSTful services (the descriptions of these HTTP service endpoints are provided [in](#_HTTP_URI_summary) HTTP URI summary section) . The services produce and consume payload of XML content format.

# **Protocol Descriptions**

## **CoAP**

A screenshot of a cell phone

Description automatically generated

Figure :CoAP over UDP stack

The sensors and actuators in the emulation environment communicate to the gateway via CoAP over UDP on their respective L4 port addresses. The CoAP interface of the gateway and the sensors and actuators in the emulator environment together constitute a Constrained ReSTful Environment (*abv .* CoRE) [1] [2]. The CoAP interface implements a resource discovery mechanism when the project starts up . The gateway sends a GET request to well known URI of each CoAP sensor or actuator - */.well-known/core* [1]. The underlying CoAP server operating in the elements of the emulator will respond with a list of available resources on the server along with the attributes available for them namely – content type: JSON, resource type: e.g. temperature , CoAP observability ,interface type: sensor, actuator etc. The resources and their attributes are returned via *Web-Link* [1] format as per RFC 6690 and RFC 7252 . The subsequent data are requested by the gateway via confirmable GET,POST methods. The nodes in the emulator environment respond to the methods with the appropriate *ACK* message containing the payload in JSON content format. Some of the nodes in the emulator environment operate via CoAP Observe mechanism [3]. The gateway initializes a Observe REGISTER to these nodes, after that these nodes send payload in their *ACK* responses whenever these nodes detect a change in representation of their resource.

Message sequence charts for different operations of gateway are shown :

CON : GET /temp

MID : a Token : x

2.05 ACK *{json-payload}*

MID : a Token : x

CON : GET /temp

MID : a + 1 Token : y

2.05 ACK *{json-payload}*

MID : a + 1 Token : x

GW

Sensor

CON : POST /foam *{json-payload}*

MID : a Token : x

2.04 ACK *{json-response-message}*

MID : a Token : x

CON : POST /foam *{json-payload}*

MID : a + 1 Token : y

2.04 ACK *{json-response-message}*

MID : a + 1 Token : x

GW

Actuator

Figure 4:CoAP GET for resource at /temp and POST for resource at /foam

OBSERVE seq : 0 GET /smoke REGISTER

2.05 ACK OBSERVE seq:1 *{json-payload}*

CON 2.05 OBSERVE seq : 2 *{json-payload}*

Token : x MID : b

Token : x MID : a

Token :x MID : a

ACK *Empty Message*

MID : b

CON 2.05 OBSERVE seq : 3 *{json-payload}*

Token : x MID : c

ACK *Empty Message*

MID : c

GW

Sensor

\*value change

\* value change

Figure :Coap Observe Mechanism for resource at /smoke

Figure 4:CoAP Observe Mechanism for resource at /smoke

## **HTTP 1.1**

## 

A screenshot of a cell phone

Description automatically generated

Figure 5:HTTP over TCP Stack

The Web browser communicates with the gateway via HTTP ReST messages [4]. The application utilizes HTTP GET and POST methods in this case. All of the HTTP resources of the gateway sends and receives payload in XML content format.

An example message sequence chart of its operation is given below

HTTP : GET /data/globalStates

200 *{xml-payload}*

HTTP:POST data/isAct

200 *{No payload}*

Browser

GW

Figure :HTTP GET and POST method description

# **Implementation Strategy**

## **IoT Gateway Architecture**

The gateway architecture has two main functionalities with regards to its interactions with nodes in the emulator environment – Resource Discovery and State Generation. With regards to its interactions with HTTP client(*Web browser*), it parses the generated state and sends it to browser for every HTTP request it receives at its HTTP server socket(*localhost:8080*) as per the description of the designated URI.

### **Resource Discovery**

2.05 ACK *{Weblink payload }* /foamExt;if=Actuator …

2.05 ACK *{Weblink payload }* /smoke; if=Sensor ;obs…

GET /.well-known/core?if=\*

GET /.well-known/core?if=\*

2.05 ACK *{Weblink payload }* /temp;if=Sensor …

GET /.well-known/core?if=\*

Actuator

Obs.S

Sensor

GW

Figure : Resource Discovery Procedure Note: Obs.S : CoAP Observer enabled Sensor

The gateway translates CoAP protocol to HTTP/1.1 and vice-versa and also makes decisions based on the use-case. When the application starts, the gateway does not know the URI’s corresponding to resources where data of various sensors and actuators in the emulator environment resides. The gateway only knows the scheme(*coap*) and authority(*localhost:{port number}*) of the various nodes in the emulator environment. So it performs a resource discovery mechanism in this CoRE [1]. As shown in the simplified figure above, the gateway sends a GET request to */.well-known/core* to all the nodes in the emulator environment. Each node responds to this GET request with a WebLink list of all the available resources residing on that CoAP Server and the associated attributes to that resource. The gateway then uses the associated attributes to recognize the different sensors, actuators, finite-state-machines(sensors with CoAP Observe enabled [3] ). The gateway then constructs the complete URI under which each of the resources reside*(scheme + authority + path) .*The gateway then interacts with each of these nodes in the respective fashion acknowledging the attributes corresponding to these resources. Resource discovery is only performed once during startup of this application.

### **State Generation**

Compute States

Browser

GW

Sensor

Actuator

Obs.Sen

GET /temp

GET /smoke Observe REGISTER

2.05 *{json-payload}*

2.05 *{json-payload}*

GET /data/globalStates

200 *{xml-payload}*

POST *{xml-permission}*

Prompt User Permission

POST /foamExt *{json-actuation command}*

Repeat Process

CON *{json-payload}*

2.05 ACK

\*Value change

Figure :Simplified Gateway Operation Note: Obs.Sen : CoAP Observer enabled Sensor

The gateway autonomously generates a state vector corresponding to the different sensor measurement values and actuator states corresponding to each room at regular time interval(~*500ms* in application).The computed state vector has the fields – temperature, flash value, smoke state, actuator state. All these fields combined with the name of the room where this sensor resides is used to compute the *state* of the room. Based on comparison of these measured values with the threshold values pre-configured are used to determine whether the room has any fire in it. Based on the values of smoke sensors ,it also has warning for excessive smoke scenarios for each room. The above simplistic message sequence chart characterizes the messages exchanged in this process. Note however that it is simplified version and does not show all the sensors and actuators in the emulator environment , but for a simplified case of one sensor, one actuator and one finite-state-machine(Observable smoke sensor) which are part of one room(Note ,a room has another luminosity/flash sensor and an alarm/guidance assistance actuator). All the sensor measurements , actuator states and smoke warning and fire presence attributes together for one room constitute a *local state*. The compilation of local state for the other rooms together constitute a *global state*. The global state and the local states are available as representations of ReSTful resources as per summary provided under [HTTP URI Summary](#_HTTP_URI_summary) section.

A screenshot of a cell phone

Description automatically generated

Figure :UML diagram with dependencies and public methods of Gateway protocol translation and decision-making functionalities

A screenshot of a cell phone

Description automatically generated

Figure :Class description with public methods of HTTP ReST endpoint implementation of gateway.

## **Sensor Actuator Emulator Environment**

**T**

**L**

**S**

**E**

**A**

**Room 1**

**T**

**L**

**S**

**E**

**A**

**Room 2**

**T**

**L**

**S**

**E**

**A**

**Room 3**

Temperature Sensor

IR Sensor

Smoke Sensor

Fire extinguisher actuator

Fire Alarm Actuator

Figure :Emulator Environment use-case scenario

The sensor-actuator emulator contains the emulated versions of a temperature sensor, smoke sensor , IR sensor , fire extinguisher actuator and a fire alarm actuator. Each of these sensors reads value corresponding to the parameter it is measuring from an environment file. An environment file exists for each of the rooms and has values of those physical parameters to measured represented by comma-separated values with each line denoting a particular instance of time. So as time moves forward ,the sensor will read next line in the environment file. The first value in the file represents the temperature value, the next value represents IR-intensity or luminosity indicator values, the final value represents the smoke state (which can take values only 0,1 or 2). Similarly an actuator writes values to the end of this file as per the actuation command given to it by the gateway and thereby we model a reduction of physical quantities.

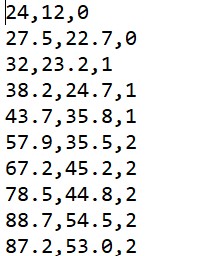


Figure :Environment file snippet

The smoke sensor is modelled as a finite state machine i.e. the gateway can register itself for a CoAP Observe [3] operation to this sensor and the sensor starts reading the physical quantity from the file but it sends the value to the gateway only when the value changes from one value to the next. So in the case of above environment file, the smoke values are sent to the gateway only when it changes from 0 to 1and from 1 to 2. The smoke sensor reads a line of the file after a fixed duration which corresponds to periodic measurements. The reason for modelling a smoke sensor this way is because an IoT enabled smoke sensor under real circumstances is operating likely on battery or some other auxiliary source of power. Responding to periodic requests from the gateway would mean spending energy over transmitting the same representation of the resource. Even practically a smoke sensor has very few set of operating states corresponding to either detection of smoke or absence of smoke, so polling the smoke sensor periodically is not as energy efficient as one might expect an IoT application to behave as. Each sensor is modelled as a resource residing on a CoAP server. Whenever this resource is invoked via any of the CoAP methods, the respective handler for it is called. In case of temperature or IR sensor the gateway periodically sends a GET request, so when the sensor receives a request it invokes its respective method handler to either read value from the environment file (in case of sensor) or write into an environment file(in case of an actuator receiving a POST message). For a smoke sensor the resource periodically reads the environment file and whenever it records a change in the state it notifies the subscribed client with the new representation for the resource.

The respective UML class relations and dependencies for the emulator environment is shown

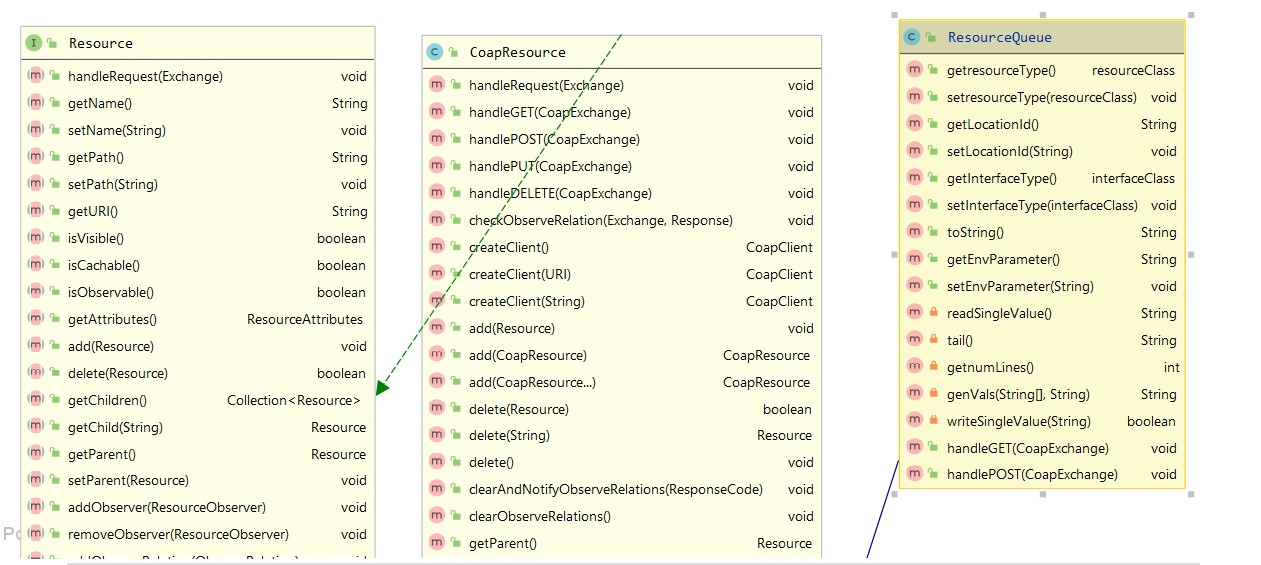
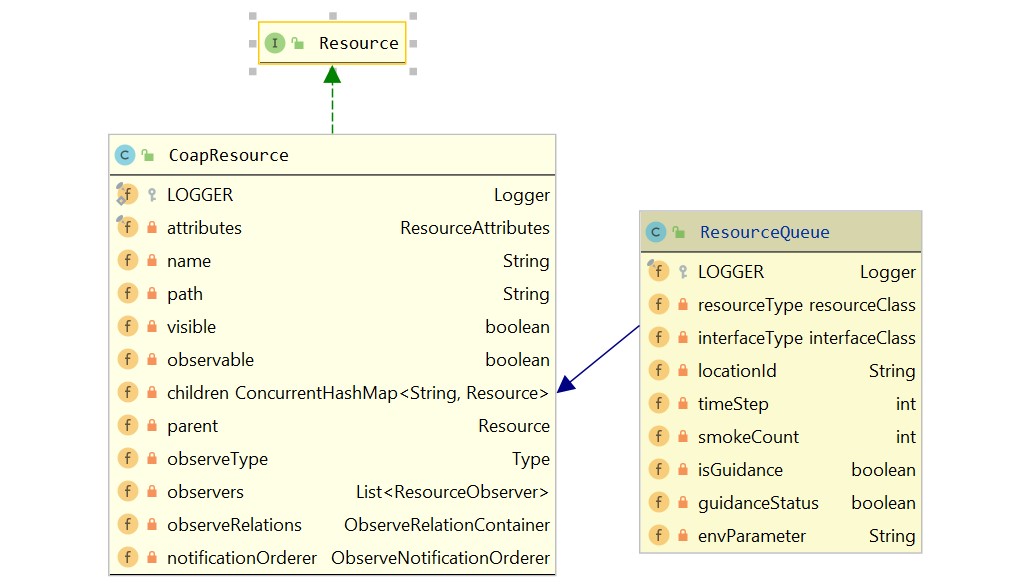


Figure :UML class diagram for extended CoAP resource used in sensor and actuators

A screenshot of a cell phone

Description automatically generated

Figure :UML class diagram for CoAP Observable resource used in smoke sensor

# **Project Setup Instructions**

## **Ubuntu**

1. Move the jar *iotservice-1.0.0.jar* to the folder where you wish to run the application.
2. The application will create a *Californium.properties* file (properties file containing CoAP stack configuration values) and *EnvironmentData* folder, so make sure the required permissions are given to the application to enable creation of necessary setup files and folders.
3. Open shell terminal.
4. Make sure that none of the ports used by the project are used by some other application running on the machine. Execute *lsof -i -P* and check with the list of ports used by this smart home application mentioned in [Appendix A](#_Appendix_A). If there are any application running on the ports used by this project , close that application by executing *kill -9 {pid-of-the application}*.
5. Navigate to the folder where you have moved the application jar file.
6. Execute *java -jar iotservice-1.0.0.jar* for the default fire in one room case, to simulate the fire in two rooms scenario execute *java -jar iotservice-1.0.0.jar version-2.*
7. After you see the application has successfully started running ,open a new browser tab and navigate to the web dashboard of the smart-home use-case at [*http://localhost:8080/console*](http://localhost:8080/console).
8. Respond to the requested prompts to observe the use case operation.

## **Windows**

1. Move the jar *iotservice-1.0.0* to the folder where you wish to run the application.
2. Open a command shell prompt and navigate to the folder where you have kept the jar file.
3. Execute *java -jar iotservice-1.0.0.jar* for the default fire in one room case, to simulate the fire in two rooms scenario execute *java -jar iotservice-1.0.0.jar version-2.*
4. After you see the application has successfully started running ,open a new browser tab and navigate to the web dashboard of the smart-home use-case at [*http://localhost:8080/console*](http://localhost:8080/console).
5. Respond to the requested prompts to observe the use case operation.

Preferably when checking between the two cases *default* and *version-2*, close the current tab in which you were viewing the results of the application and start the other version and open the web console in a new tab. This is because the JavaScript methods which were running on the browser tab will start throwing errors otherwise since it cannot reach the ReST endpoints when you stopped and restarted the application to observe results for new case. Refer [Implementation Analysis](#_Implementation_Analysis) for understanding the application logic and messages involved and their field values and parameters. Also enable JavaScript execution permissions if there were any denial policy configured into the browser which may prevent the alert messages from appearing, in course of execution. Check [Appendix A](#_Appendix_A) while using a packet sniffer to analyze the messages to configure the ports according to the correct protocol associated with the port.

# **Implementation Analysis**

## ***Default case(version-1)***

In version-1 which is the default sensor environment scenario which runs on execution, one of the rooms catches fire namely ‘Room 3’. The threshold for temperature, IR spectrum, smoke measurements to qualify as a fire situation is 58, 38 and *Intense*(corresponds to value of 2 from the environment file) respectively. The units are in a linearly scaled version of SI units and works for demonstration purposes, however these measurements may vary from real-world measurements in actual scenarios of fire breaking out in a room and should only be regarded as theoretical values for proof-of-concept purposes.

After execution, from the shell window we can see logs of various tasks done in the application as it starts up. These are identical for both the environment cases *version-1* and *version-2*.

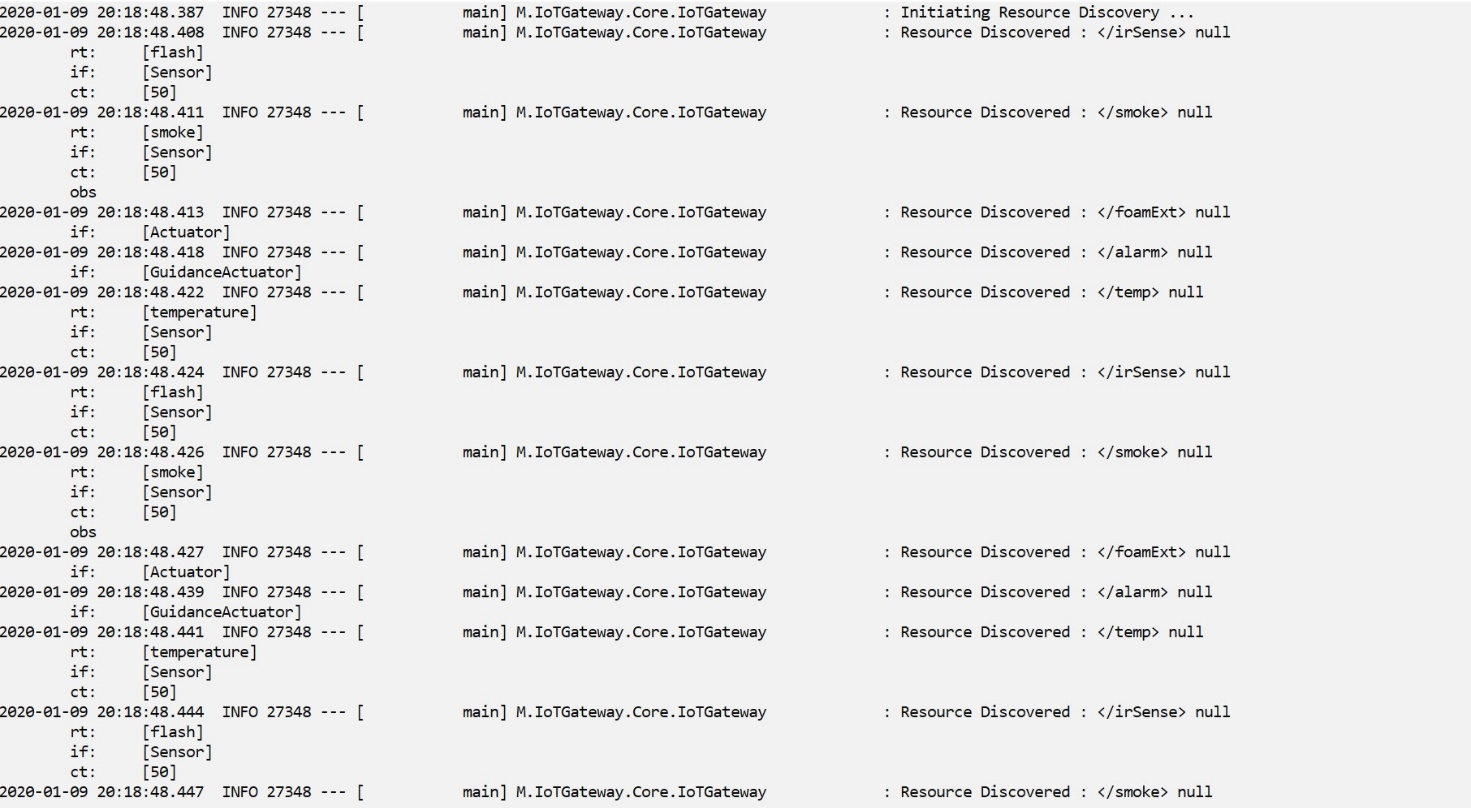
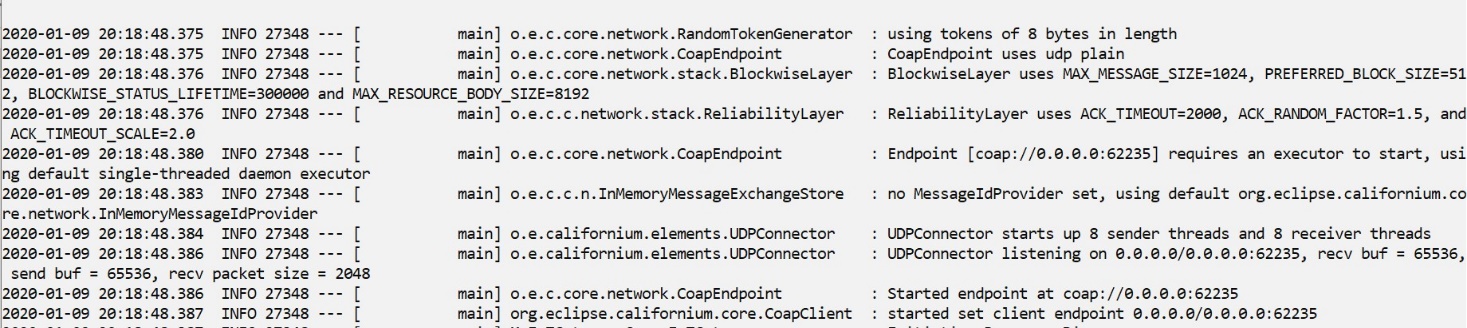


Figure :Resource Discovery logs and gateway starting at coap://localhost:62235

The web console looks like the figure below where the top row indicates the sensor measurement readings for the three rooms and the bottom row indicates states of alarms corresponding to those rooms and the flags of the gateway regarding a room’s state having presence of smoke or fire and the statuses regarding guidance actuator to the emergency exit.

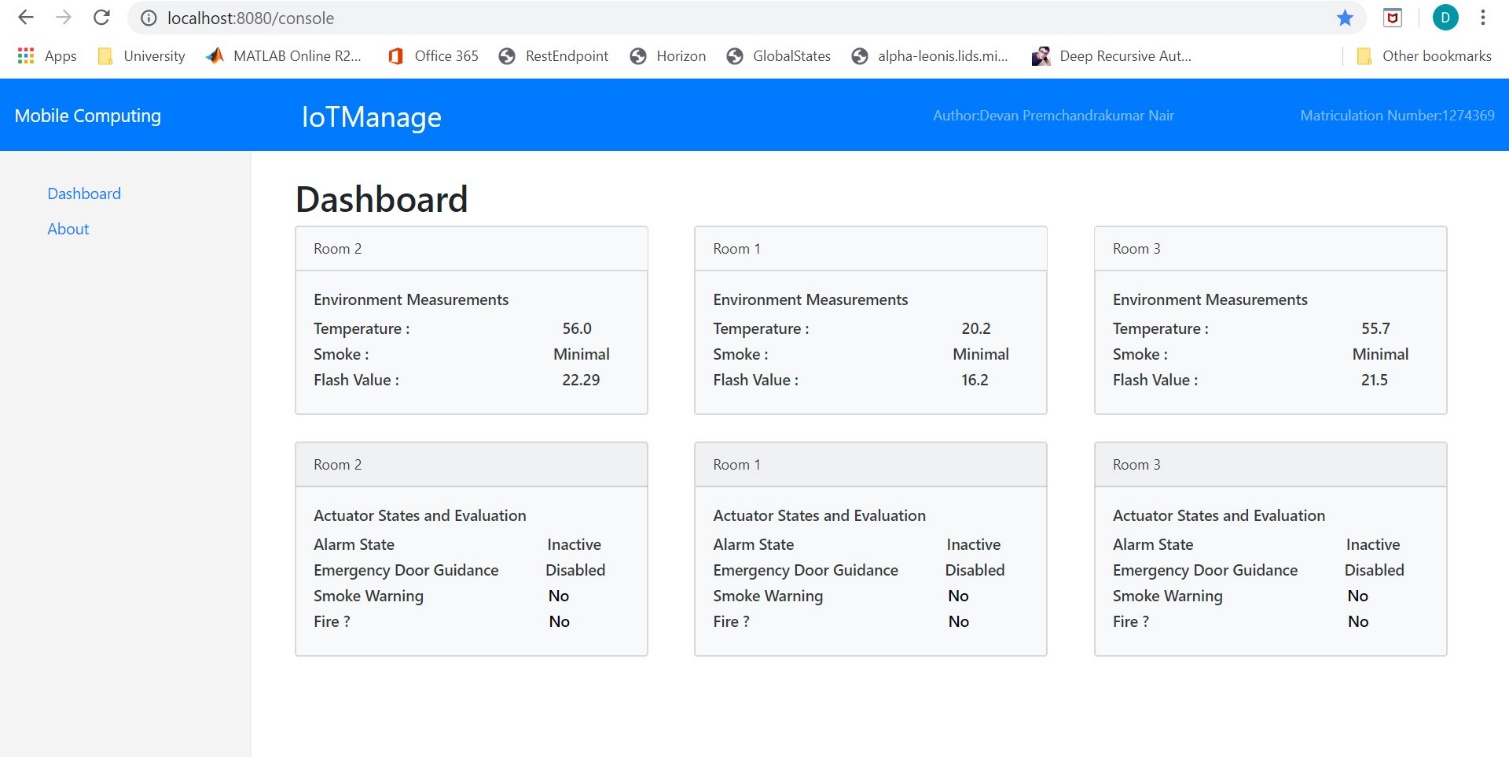
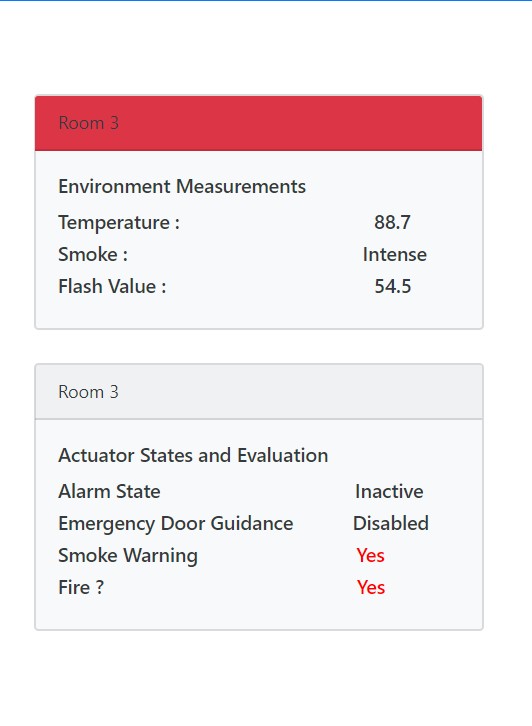
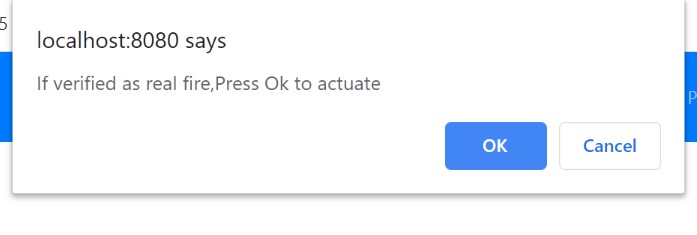
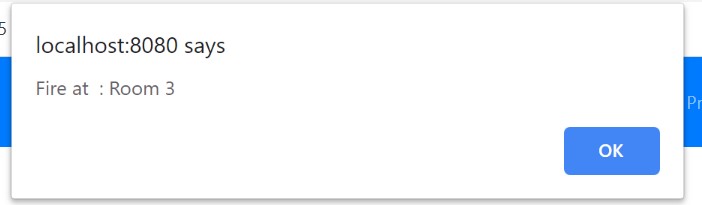
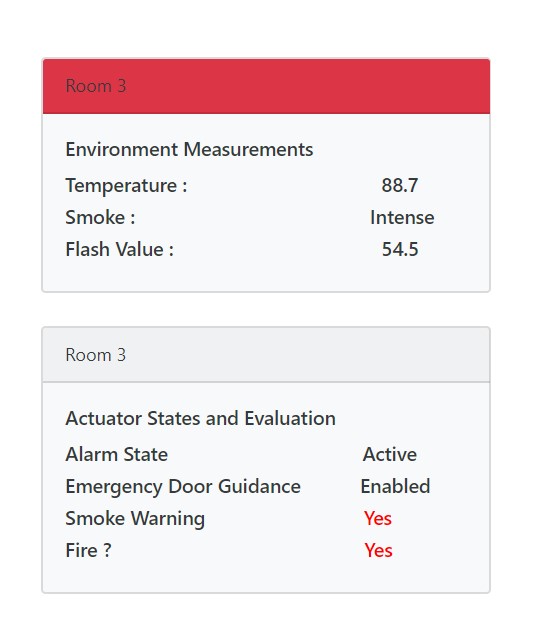


Figure :Web Console of IoT Gateway

Then we observe the environment measurement values to be changing. This is because the sensor emulators are reading parameter values from the environment file. Irrespective of the case (*version-1* or *version-2*) the parameter values for Room 1,Room 2 and Room 3 are stored in EnvironmentData folder (that was created when the project started up) under filenames ‘locn\_1.txt’, ‘locn\_2.txt’, ‘locn\_3.txt’ respectively. As time passes we see for ‘Room 3’, the temperature and IR spectrum readings increase and the room’s smoke detection transitions from *Minimal* to *Intense* and *Smoke Warning* flag is set to ‘Yes’. Eventually the measurements cross the threshold values for detection and gateway recognizes fire in ‘Room 3’ and sets the appropriate flag, so we see the *Fire ?* transition to a ‘Yes’.

At this point we receive an alert notification from the server notifying that ‘Fire at : Room 3’. This pop-up alert is meant as a model for a cell-phone alert or similar mobile-device alert mechanism as in the real world, it is highly unlikely that users of this application would be checking statuses of sensor readings from a web-browser while there is a fire developing in some other part of the house. However the alert does the minimum of notifying the user that there is a fire at a particular room and that he should check the room to make sure about the presence of fire. This verification is modelled to incorporate false positive cases of fire interpretation by the gateway due to sensor malfunctioning or some other reason, such that fire extinguishing action or alarm is not necessary to be triggered. So in the next alert message the gateway requests the user to verify if a real fire-like situation has emerged and if one presses *OK* , one observes the *Alarm* *State* and *Emergency* *Door Guidance* states transition to *Active* and *Enabled* respectively. Then we observe that the temperature and IR spectrum readings decrease until after sometime the levels fall below the threshold values and the flags in the room state observation card have *Smoke Warning* and *Fire* *?* change to *No*.



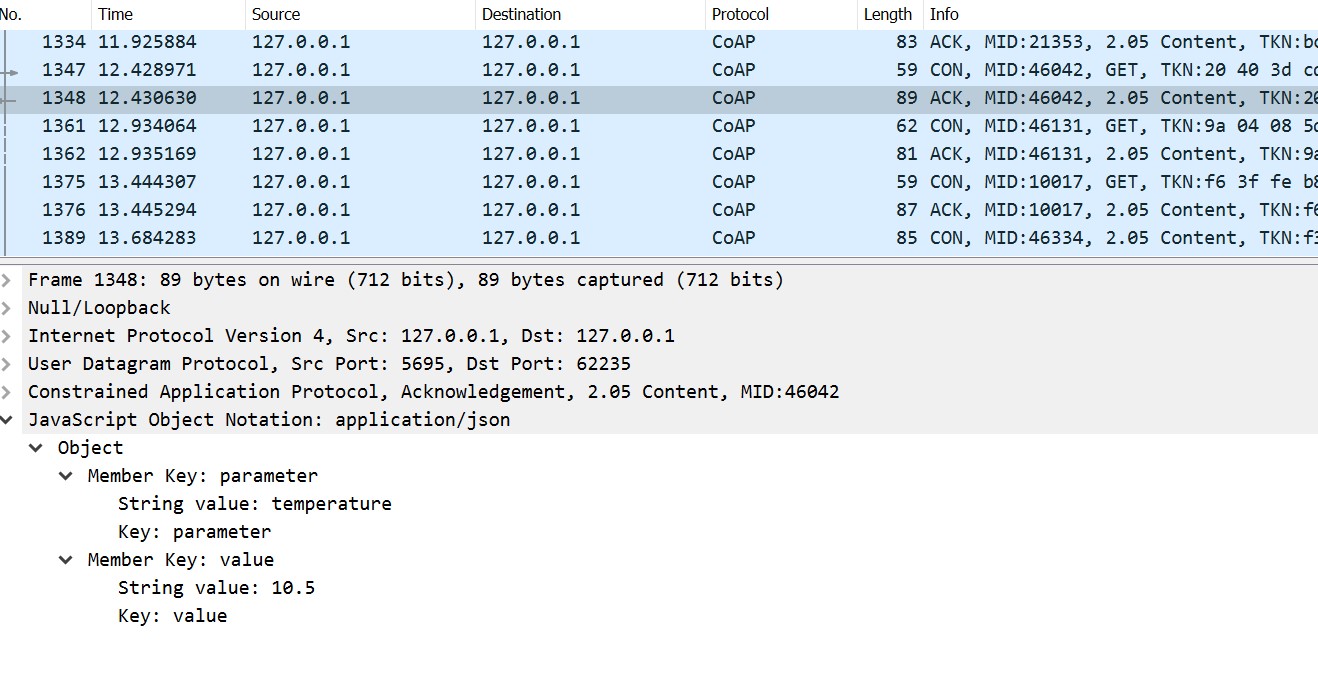
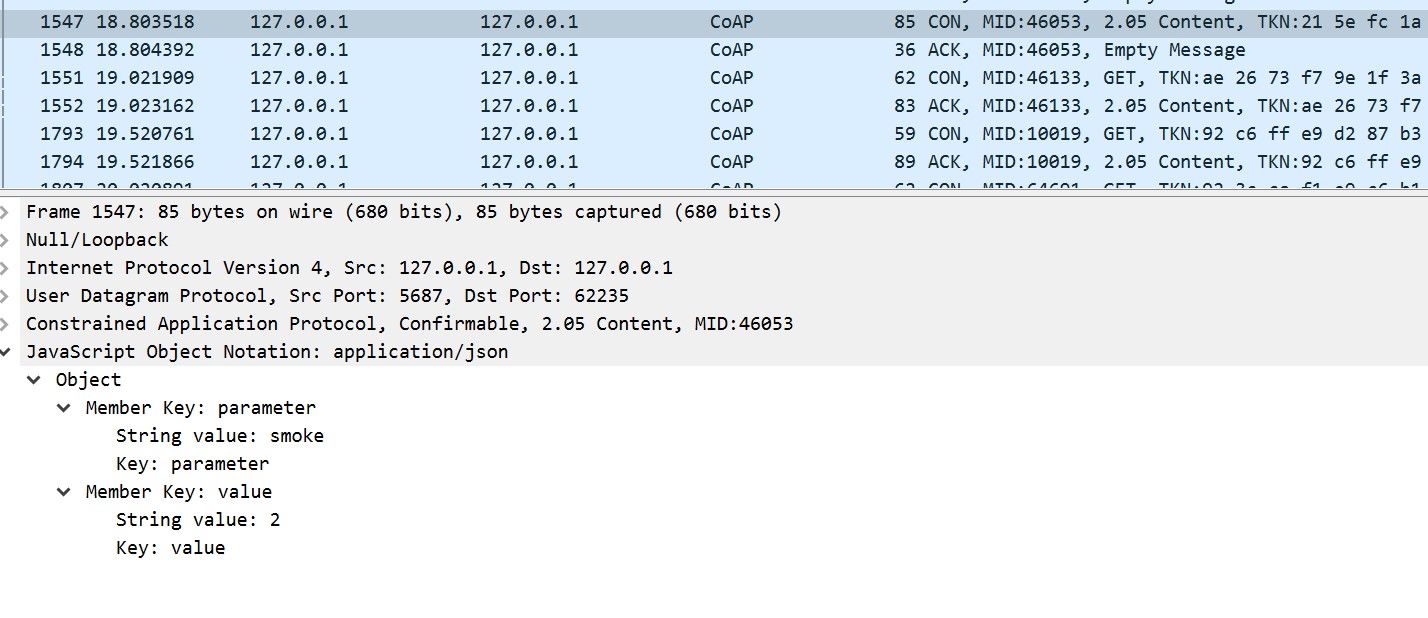
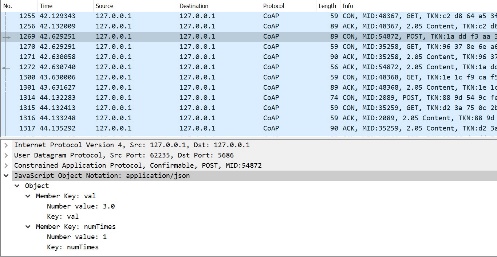
When actuation is allowed

When actuation is not allowed

Figure :Web console responses

However if user had confirmed that the fire that was detected by the gateway was false because of sensor malfunctioning(or some other scenario not qualifying to be called a fire but might have similar high measurement readings from sensor due to circumstances), one can select the *Cancel* option from the second alert window that had popped up. Then we will see the scenario as described in the bottom right image in *Figure 15*. We see the gateway faithfully displaying the sensor measurement values and its interpretation of the state of the room as per its configured logic, however the actuators and the alarms and guidance system concerned with that particular room is not triggered an they are still in *Inactive* and *Disabled* state as shown in the figure above.

Next we will have a look at the messages that were transferred between the gateway and the emulator environment, from the packet captures. The structure of these messages will be identical for *default* and *version-2* execution, as only values related to physical parameters of rooms change and all the other configuration stay as it is.



**{**

**“parameter” : “temperature”,**

**“value”:”10.5”**

**}**

Figure :Sensor Measurement CoAP messages

In the CoAP messages received from sensors above, we see the structure in which the sensors in the emulator provide a representation for the underlying resource to the gateway. The structure of these messages can be summarised as a pseudo-code as :

message

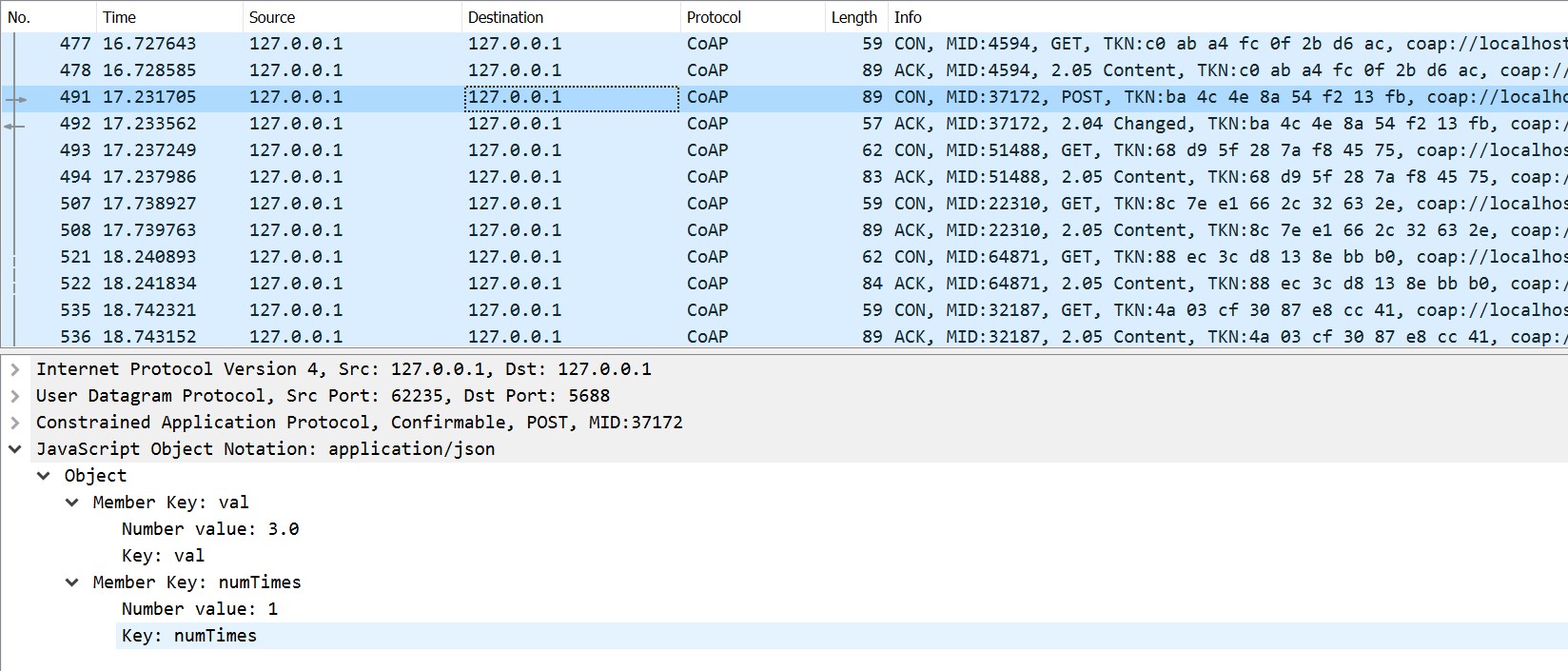
{ parameter

value

}

Note the difference however where in the top image of *Figure 16*, the sensor sends representation in *ACK* message , however in the bottom image of *Figure 16* the sensor sends its representation in a *CON* message. This is because the second image is packet trace of smoke sensor, which we have designed as a finite state machine communicating via CoAP Observe mechanism [3].

In the figure below we look at the CoAP messages sent from gateway to the actuators in the emulator environment.



**{**

**“val” : “3.0”,**

**“numTimes”:”1”**

**}**



**Response of alarm after getting activated by gateway.**

Figure :CoAP messages between gateway and actuators

Similar to previous case with sensors, we can write a pseudo-code for the actuation command sent from gateway to actuator as shown below. In these messages *val* corresponds to valve flow rate for *smart* fire-extinguisher actuator and it is commanded to operate for *numTimes* number of unit time intervals. The actuation command for guidance actuator and the alarm is the same and is a simple command asking the respective actuators to transition from an *Inactive(Disabled)* state to *Active(Enabled)* state.

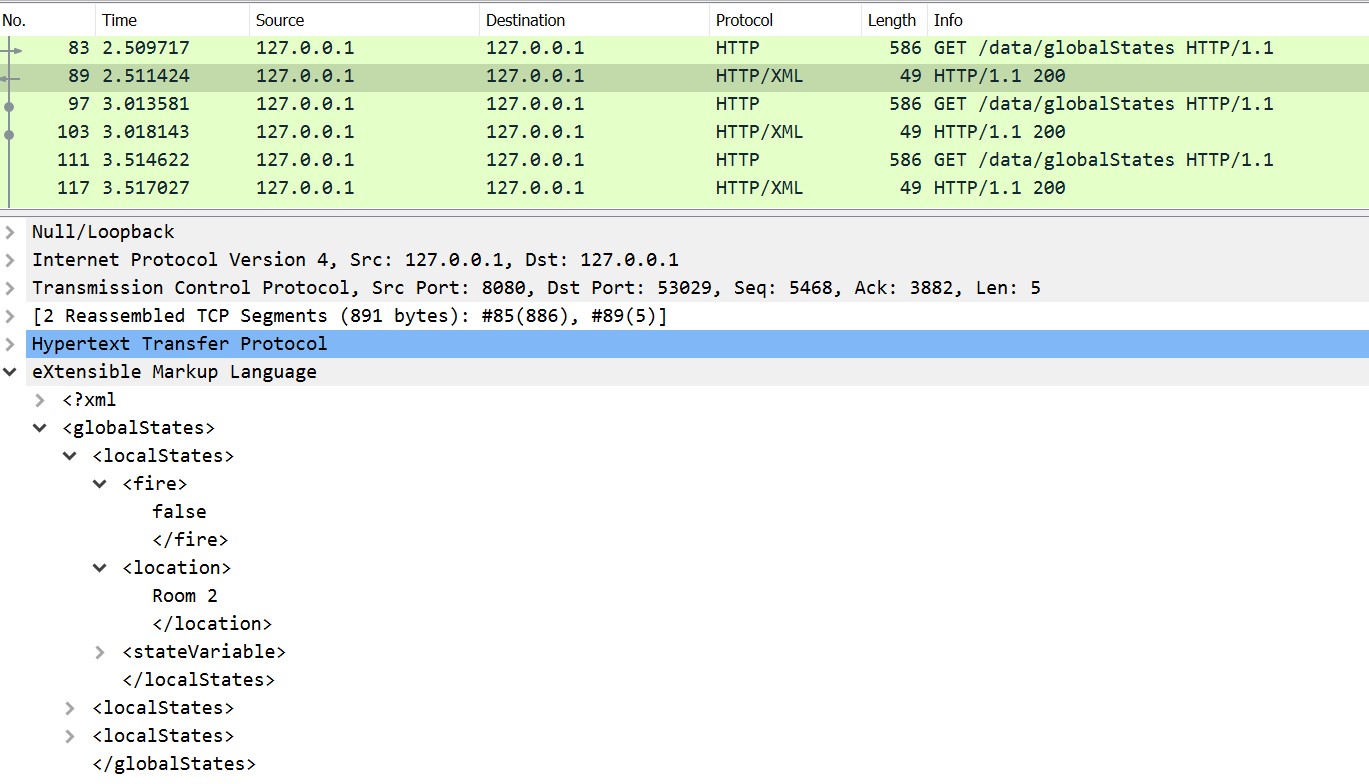
Message{

val

numTimes

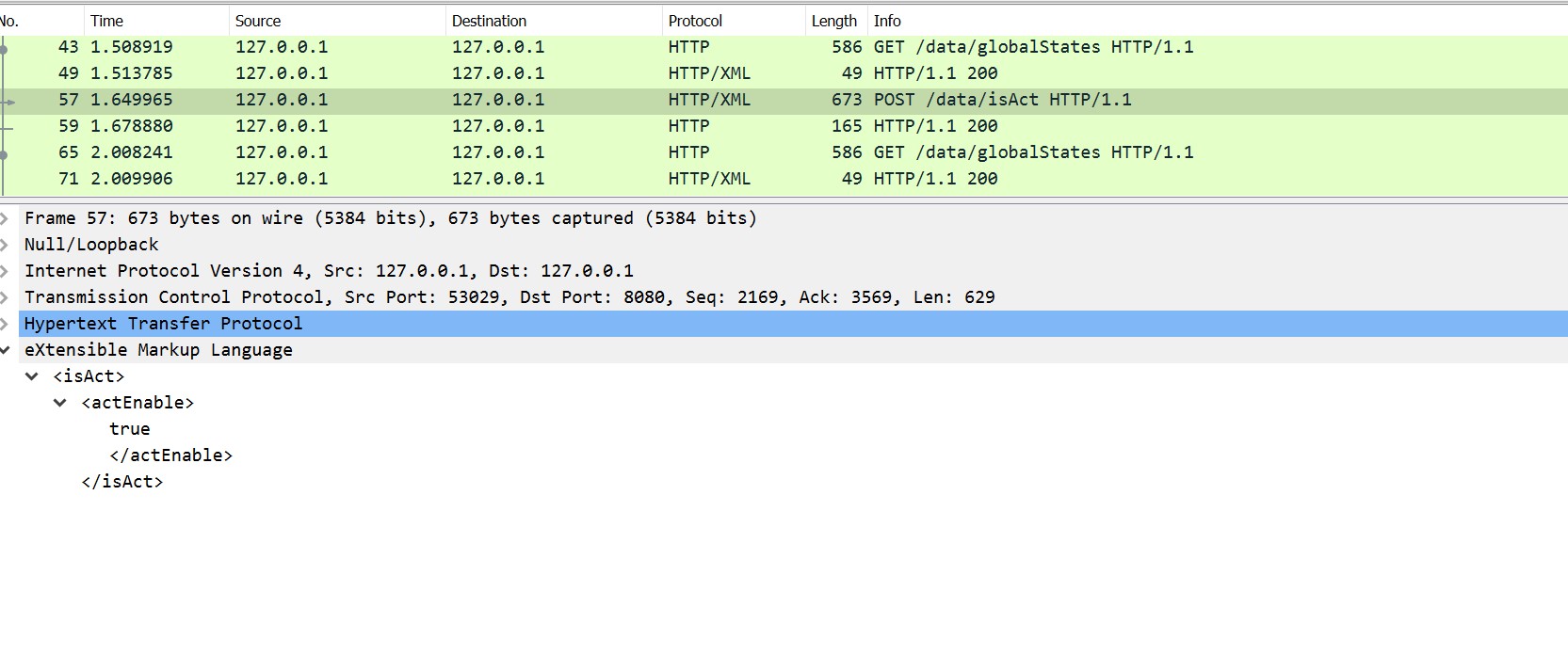
}

Finally, we take a look at the HTTP ReSTful messages sent between gateway and the browser. The application has a script running which periodically retrieves all the information regarding to sensors, actuators and their respective states from one of the ReSTful endpoints offered by the gateway(See [HTTP URI Summary](#_HTTP_URI_summary) for details). It then processes it to render necessary views and prompts.



**Full XML Response Content**

Figure :HTTP messages between gateway and browser



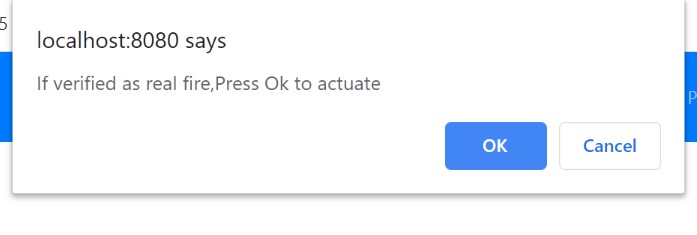
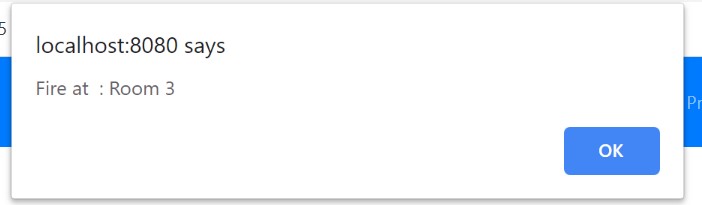
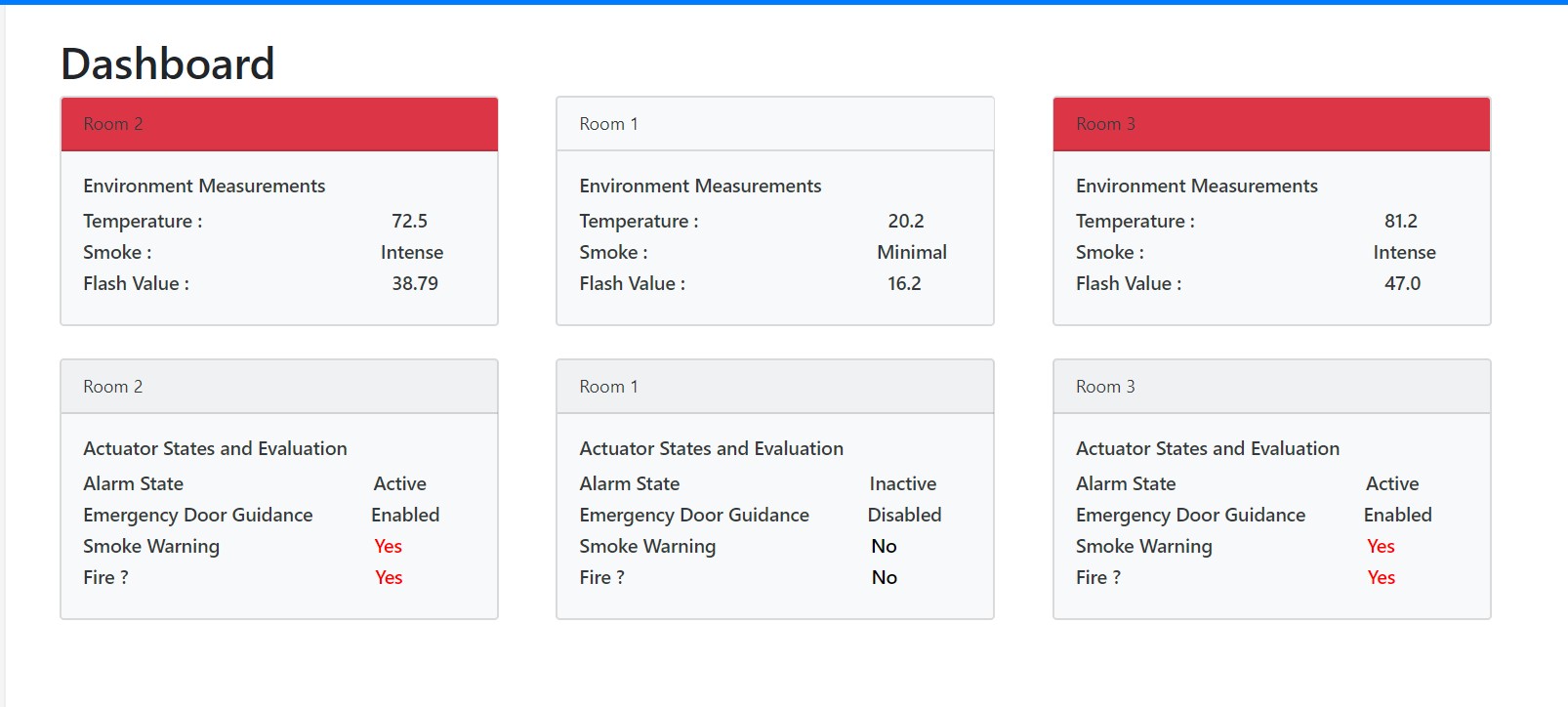
**Command to enable actuation**

Figure :HTTP POST message to send user permission regarding actuation

## ***version-2(Non-default case)***

Similar to the default case the application logic and messages transferred will stay the same. The difference lies in the different response of gateway when it detects fires in two distinct room. In this case both ‘Room 2’ and ‘Room 3’ catch fire. However the gateway detects the fire in Room 3 first ,so it displays a message identifying a fire in ‘Room 3’. However the gateway is still running in the background and detects another fire in ‘Room 2’. Now the gateway does not wait for response from user to enable actuation, as it considers it highly unlikely for two rooms to give a false identification of a fire and since the fire is happening in two separate places probably it could turn catastrophic pretty quickly. Even practically this is a highly likely scenario. It notifies on your mobile device in order to account for possibility that you may not be physically aware of the presence of fire. So irrespective of the option that you chose when the gateway prompted for actuation permission, it will have started the actuation in rooms which caught fire. It will activate the respective guidance actuators and alarms corresponding to the rooms which have caught fire. After a while we see the sensor measurement reading decreasing (no matter if we denied permission to actuate as well).

The messages transferred will be similar. The only change between the two versions is the values in the environment files, from which the sensors read the representation which has to be sent to the gateway. In *version-2*, ‘Room 2’ catches fire along with ‘Room 3’. So there will be changes in the temperature, IR spectral power measurement and smoke levels from *default* case.



Irrespective of the action, gateway will raise alarms and extinguish fire in two rooms

It detects fire Room 3 first and sends alert.

Shortly later it detects fire in Room 2,so it overrides permission procedure

Figure :Gateway response when two rooms have caught fire

# **CoAP URI Summary**

|  |  |  |
| --- | --- | --- |
| CoAP URL | Description | Resource attributes  (if Type;IsObservable?) |
| coap://localhost:5685/temp | Temperature Sensor | Sensor; No |
| coap://localhost:5686/irSense | IR Sensor | Sensor; No |
| coap://localhost:5687/smoke | Smoke Sensor | Sensor; Yes |
| coap://localhost:5688/foamExt | Fire Extinguisher actuator | Actuator; No |
| coap://localhost:5689/alarm | Alarm Actuator | GuidanceActuator; No |
| coap://localhost:5690/temp | Temperature Sensor | Sensor; No |
| coap://localhost:5691/irSense | IR Sensor | Sensor; No |
| coap://localhost:5692/smoke | Smoke Sensor | Sensor; Yes |
| coap://localhost:5693/foamExt | Fire Extinguisher actuator | Actuator; No |
| coap://localhost:5694/alarm | Alarm Actuator | GuidanceActuator; No |
| coap://localhost:5695/temp | Temperature Sensor | Sensor; No |
| coap://localhost:5696/irSense | IR Sensor | Sensor; No |
| coap://localhost:5697/smoke | Smoke Sensor | Sensor; Yes |
| coap://localhost:5698/foamExt | Fire Extinguisher actuator | Actuator; No |
| coap://localhost:5699/alarm | Alarm Actuator | GuidanceActuator; No |
| coap://localhost:62235 | IoT Gateway endpoint | - |

# **HTTP URI Summary**

|  |  |  |
| --- | --- | --- |
| URL | Description | Method |
| http://localhost:8080/console | Displays the main console dashboard of the application | GET |
| http://localhost:8080/data/globalStates | Returns the state of all sensors and actuators and rooms used in the application | GET |
| http://localhost:8080/data/Room%201 | Returns all the sensor measurement and actuator state values for Room 1 | GET |
| http://localhost:8080/data/Room%202 | Returns all the sensor measurement and actuator state values for Room 2 | GET |
| http://localhost:8080/data/Room%203 | Returns all the sensor measurement and actuator state values for Room 3 | GET |
| http://localhost:8080/data/locn | Returns list of names of rooms in the emulator environment. | GET |
| http://localhost:8080/data/isAct | Sends the user prompt response to enable or disable actuation as  <isAct><actEnable*>{user response}*</actEnable></isAct> | POST |

# **Appendix A**

|  |  |  |  |
| --- | --- | --- | --- |
| Port | Protocol | Description | Room # |
| 5685 | CoAP | Temperature Sensor | Room 3 |
| 5686 | CoAP | IR Sensor | Room 3 |
| 5687 | CoAP | Smoke Sensor | Room 3 |
| 5688 | CoAP | Fire Extinguisher actuator | Room 3 |
| 5689 | CoAP | Alarm Actuator | Room 3 |
| 5690 | CoAP | Temperature Sensor | Room 2 |
| 5691 | CoAP | IR Sensor | Room 2 |
| 5692 | CoAP | Smoke Sensor | Room 2 |
| 5693 | CoAP | Fire Extinguisher actuator | Room 2 |
| 5694 | CoAP | Alarm Actuator | Room 2 |
| 5695 | CoAP | Temperature Sensor | Room 1 |
| 5696 | CoAP | IR Sensor | Room 1 |
| 5697 | CoAP | Smoke Sensor | Room 1 |
| 5698 | CoAP | Fire Extinguisher actuator | Room 1 |
| 5699 | CoAP | Alarm Actuator | Room 1 |
| 62235 | CoAP | IoT Gateway endpoint | - |
| 8080 | HTTP | IoT Gateway web console port | - |

Easy way to check if some other application on your Linux machine is using any of the ports which are required for running this project is, execute *lsof -i -P | grep 56\*\* | grep 62235* . To observe CoAP packets in Wireshark right-click on any packet from port 62235 -> Decode As -> CoAP.

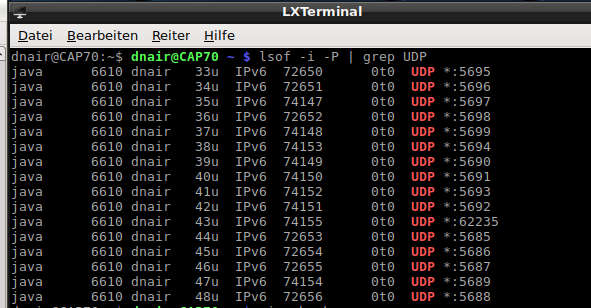


Figure :List of used ports for CoAP when the application is running

The HTML5 pages used in web-console are validated.

# **Acknowledgements**

I would like to thank Mr. Frick , Mr. Shala and Prof. Lehmann for advice and guidance provided in this project. Further I would like to thank the open-source community at large for enabling a platform for free collaboration of ideas and support provided for CoAP and HTTP ReSTful libraries.

# **References**

|  |  |
| --- | --- |
| [1] | Z. Shelby, "Constrained RESTful Environments (CoRE) Link Format (RFC6690)". |
| [2] | Z. Shelby, K. Hartke and C. Bormann, "The Constrained Application Protocol (CoAP) (RFC 7252)". |
| [3] | K. Hartke, "Observing Resources in the Constrained Application Protocol (CoAP) (RFC 7641)". |
| [4] | R. Fielding, J. Mogul, H. Frystyk, L. Masinter, P. Leach and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1 (RFC 2616)". |