**Name** : M.Tharshineeswari suki

**College code**  : 9530

**College name**  : St.Mother Theresa Engineering college

**Naan mudhalvan id** : au953021104053

**Team id :** Proj\_201055\_Team\_1

**Project Name**  : Serverless IOT data processing

**INTRODUCTION :**

Application have gained massive popularity in recent times. IoT network comp smart and connected objects called “things”. Things communicate with Each other without any human intervention and generate a significant amount of data. Things are heterogeneous and build a dynamic infrastructure. IoT devices have already outreached the human population and are expected to cross the number of 500 billion by 2030. IoT devices differ in architecture sensing capabilities and other aspects such as memory and power. IoT Network is dynamic as devices are deployed in a frequently changing environment. Furthermore, IoT devices are resources constrained as they are often Deployed with limited memory, power and processing capability.

**Problem Statement:**

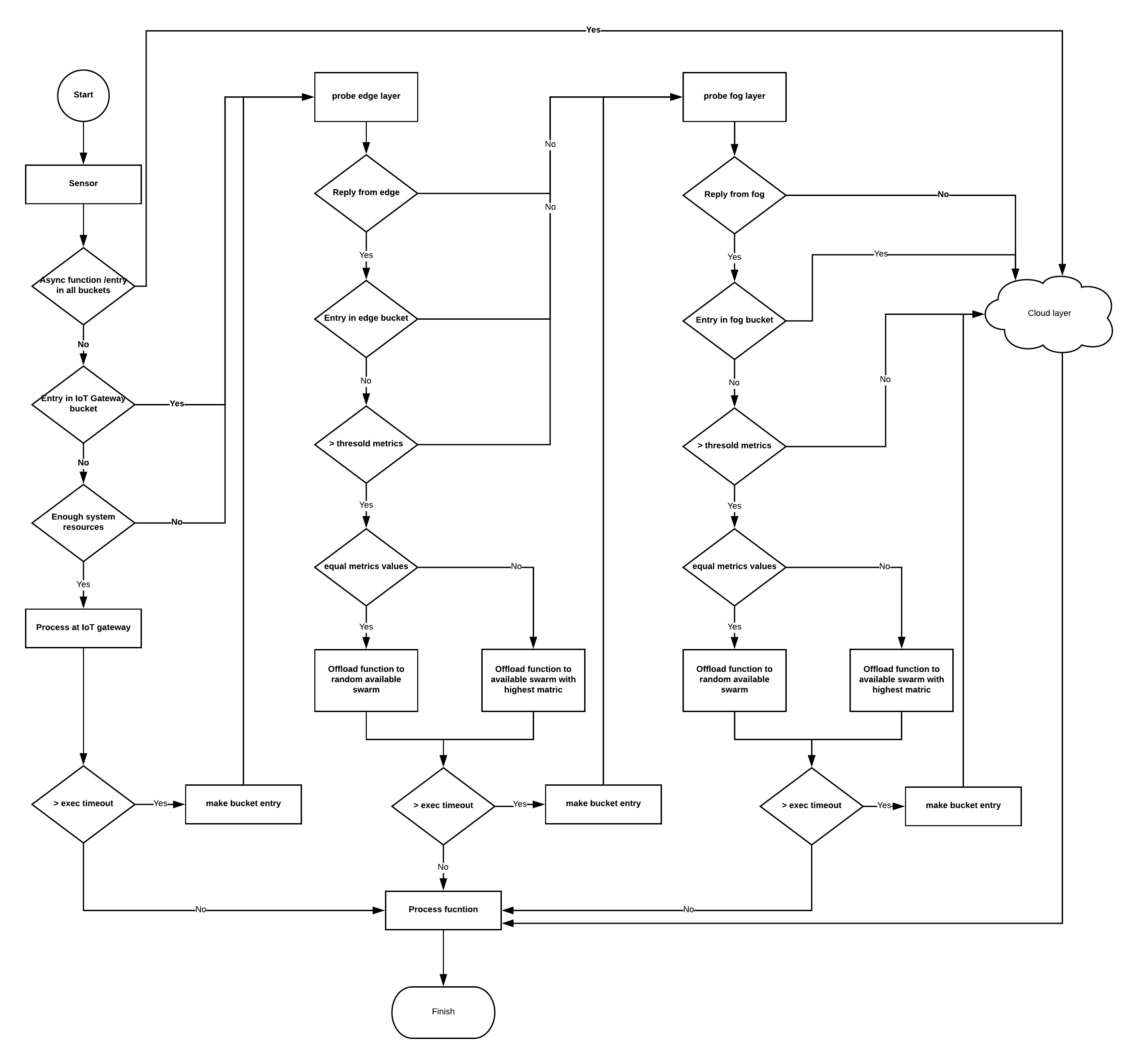
Cloud computing and IoT devices are characterized by contrasting characteristics .For instance, IoT devices work with limited capabilities whereas the cloud provides an illusion of infinite resources. Cloud computing provides the required resources to the IoT network. Due to the limitations of IoT devices, generated data is offloaded to cloud-based resources for further processing, and the cloud sends the results back upon processing the data.

IoT devices in conjunction with cloud resources perform efficient data processing. However, such solution has the following limitations:

• High latency: Offloading a small task to the cloud scenario relatively more time than processing it locally at the IoT device.

• Privacy concerns: Some tasks need more privacy, which makes it infeasible to offload their processing to the cloud.

• Support for mobility: In case of non-stationary sensing devices, it may be possible to offload processing data to the cloud. In such a , a sensor should be able to process it locally.

**Methodology: ease**

We divide the function execution offloading flow at each layer for the case of understanding.

**Offloading flow at IoT-gateway**

1. The IoT-gateway receives the data from the IoT device and triggers the function.
2. Asynchronous functions run longer than synchronous functions. IoT-Gateway offload function to cloud layer when one of the following conditions exists:
3. If the function is asynchronous.
4. If function name exists in Function bucket at each layer.
5. IoT-gateway probes function to edge layer upon encountering following Scenarios:
   1. If function name exists in the Function bucket at IoT-gateway Layer.
   2. If IoT-gateway does not have enough System resources.
6. In the case of the synchronous function and IoT-gateway has adequate system resources; it starts the execution.
7. When an OpenFaaS function execution outlives the exec timeout without giving the output, the function name is stored in the function Bucket with the current date-time stamp. Upon making the entry, IoT-gateway probes the next layer.
8. If function finishes processing within the defined value of exec timeout, Gateway finished the function processing.

**Offloading flow at edge layer :**

* 1. IoT gateway offloads the function execution to edge layer if the edge Layer sends a multicast response in answer to the probe multicast request.
  2. The function is offloaded to the next layer if:
     + 1. Current layer remains unresponsive for a defined time limit.
       2. There is an entry of function name in Function bucket at edge layer.
       3. If the metric value is lower than the threshold.

3. In case of more than one Docker swarm:

(a) if obtained metrics are equal (equal number of active nodes), the function is randomly offloaded to one of the available nodes.

(b) else, we offload the function to the swarm with highest metric.

4. If function exceeds layer’s exec timeout, a Function bucket entry is made, and function execution is offloaded to the next layer

5. else, the function is processed at the edge layer successfully.

**Offloading flow at fog layer:**

1. Function is offloaded to the fog layer if it sends a valid multicast reply.

2. The function is offloaded to the next layer if:

(a) current layer remains unresponsive for a defined time limit.

(b) there is an entry of function name in Function bucket at fog layer.

(c) If the metric value is lower than the threshold.

3. In case of more than one Docker swarm:

(a) if obtained metrics are equal (equal number of active nodes), the function is randomly offloaded to one of the available nodes.

(b) else, we offload the function to the swarm with highest metric.

4. If function exceeds layer’s exec timeout, a Function bucket entry is made, and function execution is offloaded to the next layer

5. else, the function is processed at the fog layer successfully.

**Offloading flow at cloud layer :**

1. Upon receiving the function execution offloading, cloud layer processes

the function and returns the result.

**Software Requirements:**

To implement the solution, there are some software requirements needed. The

Requirements are as follows

* Apache Open Whisk.
* IoT-Gateway.
* OpenFaas.
* Kuberless platform.

**Innovation:**

Although initially proposed for the cloud, serverless computing has also found its place on Internet of Things (IoT) while bringing functions closer to the devices, in order to reduce latency and avoid unnecessary energy and resource consumption.

**IBM cloud based setup:**

IBM cloud-based setup is a paid service provided by IBM cloud as IBM functions. IBM cloud function uses the only cloudant as a database. Furthermore, a command line utility is available for users to deploy, modify and execute functions. The utility for the local setup can be cloned and Installed form the Github project. IBM cloud’s command line utility also provide similar functionality and can be download from IBM cloud webpages.

**Functions Involved:**

* Apache OpenWhisk python function

import smtplib

from email.mime.multipart import MIMEMultipart

from email.mime.text import MIMEText

def main(dict):

fromaddr = ‘sender@gmail.com’

toaddr = ‘receiver@outlook.com’

. msg = MIMEMultipart()

msg[‘From’] = fromaddr

msg[‘To’] = toaddr

msg[‘Subject’] = ‘Apache OpenWhisk’

body = ‘TEST’

msg.attach(MIMEText(body, ‘plain’))

server = smtplib.SMTP(‘smtp.gmail.com’, 587)

server.ehlo()

server.starttls()

server.login(fromaddr, ‘password’)

text = msg.as\_string()

server.sendmail(fromaddr, toaddr, text)

return {“Status”, “Success”}

* Faas python function

import requests

import json

def handle(req):

result = {“found”: False}

json\_req = json.loads(req)

r = requests.get(json\_req[“url”])

if json\_req[“term”] in r.text:

result = {“found”: True}

print json.dumps(result)

**GitHub Link:**

**https://github.com/Mdevi10/Cloud-Computing**