

# Internet of Things realized using XINU Operating System

Ramona Vaz  
CISE Dept  
University of Florida  
rvaz@ufl.edu

*Abstract*—In today's world we find that the demand for smart devices is surging. Smart cars, smart homes, smart cities are now, reality. This paper delves into depths of how anyone with the right components and right technological frameworks could build an end to end IoT application. The focus here is on the realization of an intelligent prototype based on the Cloud Edge Beneath[8] architecture designed to provide an overview of how XINU Operating System deployed on the BeagleBone Black can be tailored to fit into today's IoT domain's changing dynamics. The system has been designed to be effectively scalable with a composite nature accommodating both analog and digital sensors and actuators. The end user of the application can remotely read the system state which in our context is the ambient temperature reading as read by the analog temperature sensors(TMP36)[2]. This interaction with the end user is smartly used to trigger system changes. For instance, below a certain temperature threshold, the LED is triggered to glow. This is accomplished by having a 3 tier architecture consisting of the sensor platform where all sensors and actuators are connected to the BeagleBone Black, the edge data processing layer and a cloud application.

## I. INTRODUCTION

The sheer number and range of sensors predicted to be connected to the cloud in the near future will definitely cause a strain on the resources of the cloud. In order to avoid this scenario there is a pressing need to have an ecosystem and scalable architecture in place to manage the ever-expanding Internet of Things. The Cloud Edge Beneath(CEB)[8] architecture proposes to establish this very paradigm. Some

of the salient features of the Cloud Edge Beneath architecture are:

- Separating Device Integration from Service/Application Development[8]:  
There is a need to integrate any kind of sensor/device through an automated process which would then cause that device to be controlled through an interface specific to that device.
- Mandate that Programmability be a Federated Service/Application Development Paradigm[8]:  
Programmers that lack domain knowledge should only be given access to utility services of sensors to develop their applications based on their requirements. There is a need to define all actor's roles in the ecosystem while maintaining inclusiveness to enable rapid proliferation.
- Optimization enabling to tackle problems of traffic and energy efficiency[8]:  
The Edge layer[6] is empowered to conduct most of the sensor data processing by being connected to sensor networks instead of sensor networks directly being queried and controlled by cloud applications.

### A. Significance of the problem

Through the deployment of Cloud Sensor Systems[5][8] our "dream" of realizing the smart city concept, bridging the gap between supply and demand while supporting energy

sustainability eventually leading to better Quality of Life and enhanced user experience is now "reality". In the realization of this "dream" there are myriad problems that need to be overcome. The onus is on us to solve these problems:

- How can we support open-ended, friction-free, continuous integration of sensors and devices into the cloud?
- How can we program the cloud-sensor system? Do we have an ecosystem that can unleash powerful application development?
- How can we use the cloud wisely? How can we keep the cloud economically scalable?
- How to sustain high system dependability despite sensor energy limitation?

## B. Technical Challenges

1) *Cloud Scalability Challenge:* Extensive external interactions between cloud services and the physical sensors is the primary cause for worry. If sensors push data once every minute, then millions of sensors will produce billions of sensor-cloud interactions, daily. This results in tremendous processing power, memory resources and huge incoming/outgoing cloud traffic. As a result, the cloud economies of scale per sensor will not stand. That is, even though the Cloud is elastic, its auto-scaling rate will be cost-prohibitive, if it is left unrestrained.

2) *Sensor Energy Challenge:* Sensor devices have limited energy capabilities. In smart city scenarios, a sensor may be queried by multiple independent cloud applications each of which requires periodical readings of the sensor. Ultimately sensor energy rapidly depletes causing unreliable and unavailable sensor-based services. Without involving the applications in the optimization problem, minimizing sensor energy consumption will remain blind-sided and limited.

## C. The XINU IoT Prototype

The development of the end-to-end IoT application using the XINU operating system deployed on the BeagleBone Black is in essence mirroring the CEB architecture sans the ATLAS components[8] as depicted in Figure 1.

1) *Making the BeagleBone Black a sensor platform:* This was achieved using components such as the BeagleBone Black with XINU as the operating system and sensors such as the temperature sensor TMP36, a tactile switch and an LED. An appropriate abstraction was implemented for the sensor representations so that sensor data could be read at edge and cloud levels using UDP(User Datagram Protocol) as the communication protocol.

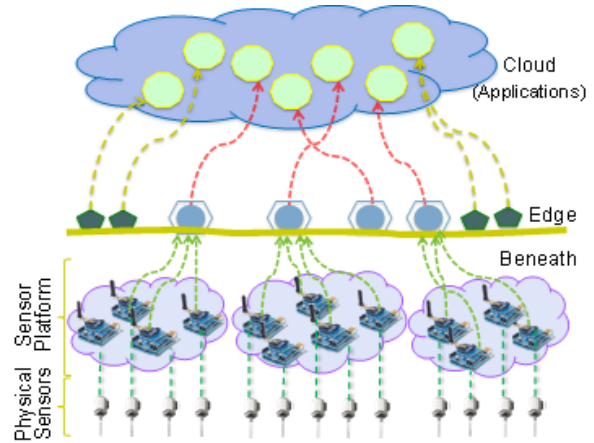


Fig. 1. The three-tier IoT architecture of the Cloud-Edge-Beneath.

2) *Edge data processing:* The edge layer comprised of a TP-Link N300 Wireless Wi-Fi Router on which a static IP address for the BeagleBone Black was registered and a standalone client-server python application that communicated with the BeagleBone Black and dumped incoming sensor data from the BeagleBone Black to a MongoDB instance hosted on mLab.com using Database-as-a-Service. The python program also retrieved data from the same MongoDB instance to which the cloud application dumped data to.

3) *Cloud application:* A hybrid application with speech recognition and text to speech

capabilities (using Apache Cordova plugin) developed on Ionic Framework for android users which uses AngularJs for all the heavy-lifting like making REST[1] calls to the Node.js application hosted on Heroku made up the cloud application layer. The Node.js application on Heroku which is a cloud Platform-as-a-Service was used as the driver for the MongoDB connection hosted on mLab.com using Database-as-a-Service.

## II. RELATED WORK

The Gator Tech Smart House (GTSH) an experimental facility for research on independent living for elders at the University of Florida is built on the Cloud Edge Beneath(CEB) architecture which uses ATLAS components[4].

## III. DESIGNING THE PROTOTYPE

The design of the prototype is based on the Cloud Edge Beneath Architecture[8]. The 3 tiered architecture is adhered to while exploring open source technologies and frameworks to replace ATLAS components[4].

### A. Device Driver Implementation for XINU I/O

Designing and implementing a device driver for a thing in the Internet of Things (IoT) is different from what is traditionally done in a computer system. This is because in the latter case, the objective is to allow processes in an operating system to access and utilize devices that are either connected directly to the computer, or remotely through the network. In an IoT, things should utilize and use each other[7]. Hence, both processes internal to a thing (For example, Xinu processes running on a Beagle Bone Black platform) as well as other nearby things ( other Beagle Bone Black(BBB) or Dragon platforms) should be able to access and utilize each other as devices. Here, we use device and thing loosely interchangeably. Users of these device drivers would be Xinu processes running on BBB as well as other devices and things external to the BBB. The

traditional high-level I/O interface open, close, getc, putc, read, write, cntrl will need to be revisited for Xinu processes accessing things on the BBB. Points to ponder about are elaborated below.

- In which way does the high-level interface need to change?
- What are the new concerns?
- What are the needed I/O operations?
- Are there any new operation abstractions that need to be introduced and supported?
- Would the high-level I/O interface (even the one you would design for Xinu processes) be appropriate for other external things and devices? Or the same physical device should have multiple interfaces? One for local access by Xinu processes, and others for access by external things?

### B. Device Externalization to the Edge and the Cloud

Initializing a device driver now should setup whatever is necessary within Xinu to enable Xinu processes to utilize the new drivers. Additionally, initializing a device entails optionally externalizing this device to other platforms, specifically, an edge computer[6] (also known as a gateway), and the cloud.

To understand the rationale behind this externalization process, we recall the Cloud-Sensor system architecture(Fig 1) in which a device or a thing may create a representation for itself on a an edge device co-located within the same smart space, or even with the cloud. This architecture enables application development and allows for powerful optimizations of the cloud resources and the device energy use. It is also a communication architecture in which heterogeneous devices and things may communicate with each other through an edge or the cloud.















































## IV. SYSTEM MODEL

The system model is elaborated in Figure 2. Each layer of the CEB architecture is demarcated while giving a glimpse of the technolo-

### A. Sensor Platform and the Edge

### 1) XINU I/O Interface:

- The low level device driver functions that manipulate the switch table are `adcread`, `adcinit`, `syscontrol`, `adchandler`.

- | P8             |   |                | P9                |   |                      |
|----------------|---|----------------|-------------------|---|----------------------|
| 1              | 2   |                | 1                 | 2   |                      |
| <b>GND</b>     |    | <b>GND</b>     | <b>GND</b>        |    | <b>GND</b>           |
| GPIO1_6 P8.3   |    | P8.4 GPIO1_7   | <b>VDD 3.3V</b>   |    | <b>VDD 3.3V</b>      |
| GPIO1_2 P8.5   |    | P8.6 GPIO1_3   | <b>VDD 5V</b>     |    | <b>VDD 5V</b>        |
| GPIO2_2 P8.7   |    | P8.8 GPIO2_3   | <b>SYS 5V</b>     |    | <b>SYS 5V</b>        |
| GPIO2_5 P8.9   |    | P8.10 GPIO2_4  | <b>PWR_BUTTON</b> |    | <b>SYS_RESETN</b>    |
| GPIO1_13 P8.11 |    | P8.12 GPIO1_12 | GPIO0_30 P9.11    |    | P9.12 GPIO1_28       |
| GPIO0_23 P8.13 |    | P8.14 GPIO0_26 | GPIO0_31 P9.13    |    | P9.14 GPIO0_18       |
| GPIO1_15 P8.15 |    | P8.16 GPIO1_14 | GPIO1_16 P9.15    |    | P9.16 GPIO1_19       |
| GPIO0_27 P8.17 |   | P8.18 GPIO1_2  | GPIO0_5 P9.17     |   | P9.18 GPIO0_4        |
| GPIO0_22 P8.19 |  | P8.20 GPIO1_31 | GPIO0_13 P9.19    |  | P9.20 GPIO1_12       |
| GPIO1_30 P8.21 |  | P8.22 GPIO1_5  | GPIO0_3 P9.21     |  | P9.22 GPIO0_2        |
| GPIO1_4 P8.23  |  | P8.24 GPIO1_1  | GPIO1_17 P9.23    |  | P9.24 GPIO0_1        |
| GPIO0_1 P8.25  |  | P8.26 GPIO1_29 | GPIO0_21 P9.25    |  | P9.26 GPIO0_14       |
| GPIO0_22 P8.27 |  | P8.28 GPIO0_24 | GPIO0_19 P9.27    |  | P9.28 GPIO0_17       |
| GPIO2_23 P8.29 |  | P8.30 GPIO2_25 | GPIO3_15 P9.29    |  | P9.30 GPIO3_16       |
| GPIO0_10 P8.31 |  | P8.32 GPIO0_11 | GPIO3_14 P9.31    |  | <b>VDD_ADC(1.8V)</b> |
| GPIO0_9 P8.33  |  | P8.34 GPIO2_17 | AIN4 P9.33        |  | <b>GNDA_ADC</b>      |
| GPIO0_8 P8.35  |  | P8.36 GPIO2_16 | AIN6 P9.35        |  | P9.36 AIN5           |
| GPIO2_14 P8.37 |  | P8.38 GPIO2_15 | AIN2 P9.37        |  | P9.38 AIN3           |
| GPIO2_12 P8.39 |  | P8.40 GPIO2_13 | AIN0 P9.39        |  | P9.40 AIN1           |
| GPIO2_10 P8.41 |  | P8.42 GPIO2_11 | GPIO20_20 P9.41   |  | P9.42 GPIO2_7        |
| GPIO0_8 P8.43  |  | P8.44 GPIO0_9  | <b>GND</b>        |  | <b>GND</b>           |
| GPIO2_6 P8.45  |  | P8.46 GPIO2_7  | <b>GND</b>        |  | <b>GND</b>           |

- High level driver implementation:  
Device Description Language (DDL) is a language to describe devices. The complete device driver file has been designed to be dynamically generated based on the DDL JSON file input to the stand-alone Java application.  
The Java Application reads the DDL JSON file, defines the scope of the device driver (for each type Analog, Digital; Input, Output) and stitches in only the required low level read/write functions as the cases in the switch statements of the high level read/write func-

```

{
  "ddl": {
    "device": [{
      "name": "LED_rgb",
      "pin": [{
        "id": "gnd_v",
        "index": "0",
        "type": "D",
        "mode": "out",
        "level": "high"
      }]
    }], {
      "name": "power_btn",
      "pin": [{
        "id": "pow_btn",
        "index": "7",
        "type": "D",
        "mode": "in",
        "level": "0"
      }]
    }, {
      "name": "temp_sensor",
      "pin": [{
        "id": "temp_sen",
        "index": "5",
        "type": "A",
        "mode": "in",
        "level": "0.899"
      }]
    }
  ]
}

```

Fig. 5. DDL

tions. Java Collections like ArrayLists and HashMaps, JSONObject Class, and a few custom classes tailored to manipulate the expected JSON structure have been extensively used in the logic implementation to achieve driver code generation. At the device level, the individual sensors and actuators connected to the BBB are blanketed in the abstraction layer provided. The broad categories identified are Analog and Digital, and each are further categorized as Input and Output. Considering that the Prototype has only one active BBB, there was no pressing need for an abstraction layer for the communication between the BBB and the edge device, however the UDP interaction can have the data packaged with the device ID to be scalable and still have the capability of handling the intricacies of the code and logic.

## 2) Edge Functions:

- UDP(User Datagram Protocol) server registration is done with the IP and listening port of the Edge device as the parame-

ters. In our case, the Edge device is a personal computer running a Python UDP server/client. For the UDP client registration, there is no need to specify the IP and listening port.

- There is no direct interaction between the BBB and the cloud. The Edge device mediates all interactions with the BBB.
- The Python server/client doubles up as a MongoDB connector and reads UDP packets from XINU and the cloud application and extracts JSON and writes to the Collection defined as SENSOR-COLLECTION in MongoDB.

## B. Cloud Application Implementation

1) *Underlying Frameworks:* The Open Source technologies used are

- MEAN Stack(as depicted in Fig 6)
  - MongoDB is the leading NoSQL database, empowering businesses to be more agile and scalable.
  - Express is a minimal and flexible Node.js web application framework, providing a robust set of features for building single and multi-page, and hybrid web applications.
  - AngularJS lets you extend HTML vocabulary for your application. The resulting environment is extraordinarily expressive, readable, and quick to develop.
  - Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications.
- Node.js application hosted on Heroku
- mLab providing Database-as-a-service

2) *User Interface and Features:* A hybrid application called "HEATHER" with speech recognition and text to speech capabilities (using Apache Cordova plugin) developed on Ionic Framework for android users which uses AngularJs for all the heavy-lifting like making REST calls to the Node.js application hosted on Heroku.

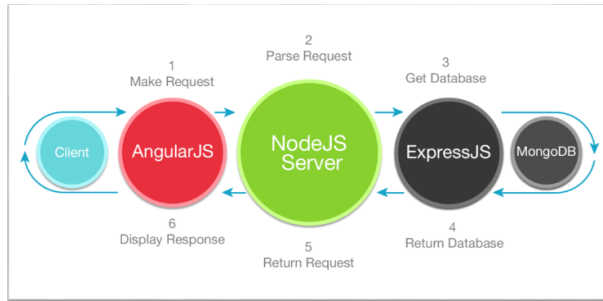


Fig. 6. MEAN Stack

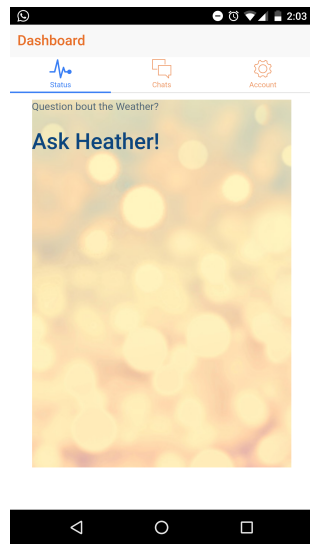


Fig. 7. Heather

We are using Node.js as the driver for the MongoDB connection hosted on mLab.com using Database-as-a-Service. The application on invocation of the tab "CHATS" (in Fig 7) prompts the user to ask for the weather. Thereafter, the REST GET[1] call is invoked to fetch the temperature which is checked against a threshold value which in our case is 26 deg Celsius and writes a value of "1" using POST[1] to the LED-STATUS field in the DB which makes the LED glow when the temperature dips beneath the threshold at the temperature sensor connected to the Beagle-Bone Black. In essence, this completes the loop from the user facing application to the sensor platform using the Edge as an intermediate. The frameworks chosen fit well together and



Fig. 8. Interface that makes the REST calls

has validated that this architecture supports a gateway application that can be expanded across platforms with a scalable nature in terms of number and types of devices (sensors and actuators).

## V. FUTURE SCOPE AND CONCLUSION

This paper has been very elaborate in detailing the implementation of an IoT application based on a specific Operating System(XINU), a specific Development Platform(System on a Chip) which is the BeagleBone Black and a few select sensors (LED, tactile switch, TMP36). These components could be swapped with a different set of components to achieve different functionalities based on specific requirements. Usage of the MEAN stack helps in rapid development and prototyping and supports JSON. In this prototype we are essentially, reducing sensor energy depletion and achieving cloud scalability with the deployment of the EDGE device while adhering to the idea of the CEB architecture.

## VI. ACKNOWLEDGMENT

I'm extremely grateful to Dr. Sumi Helal(University of Florida) for introducing me to the concept of Cloud Edge

Beneath(CEB)[8]. This project draws inspiration from the CEB architecture proposed and published by him.

#### REFERENCES

- [1] AL-FUQAHA, A., GUIZANI, M., MOHAMMADI, M., ALEDHARI, M., AND AYYASH, M. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Commun. Surv. Tutorials* 17, 4 (2015), 2347–2376.
- [2] ANALOG DEVICE. Low Voltage Temperature Sensors TMP35/TMP36/TMP37. *Analog Device* (2008), 1–2.
- [3] G, D. R., COURT, C., AND JOSE, S. Technical Reference Manual. 1–160.
- [4] KING, J., BOSE, R., AND YANG, H. I. Atlas: A Service-Oriented Sensor Platform Hardware and Middleware to Enable Programable Pervasive Spaces. {...}, *Proc. 2006 31st {...}* (2006), 630–638.
- [5] MIORANDI, D., SICARI, S., DE PELLEGRINI, F., AND CHLAMTAC, I. Internet of things: Vision, applications and research challenges. *Ad Hoc Networks* 10, 7 (2012), 1497–1516.
- [6] SHI, W., CAO, J., ZHANG, Q., LI, Y., AND XU, L. Edge Computing : Vision and Challenges. 1–10.
- [7] WANT, R., SCHILIT, B. N., AND JENSON, S. Enabling the internet of things. *Computer (Long. Beach. Calif.)* 48, 1 (2015), 28–35.
- [8] XU, Y., AND HELAL, A. Scalable Cloud-Sensor Architecture for the Internet of Things. *IEEE Internet Things J.* 3, 3 (2016), 285–298.