ISSN:0975-9646

Lightweight Platform for Internet of Things with support for CoAP Block-wise Transfer

Namrata Pawar, Madhuri Wakode

Department of Computer Engineering
Pune Institure of Computer Technology, Pune, India

***Abstract*— To make Internet of Things Platform lightweight, there is need of lightweight software and hardware components. CoAP(Constrained Application Protocol) is low power consumption and low energy communication protocol for constrained environment. We built the lightweight IoT platform using CoAP. We also enhance the platform with CoAP Block-wise transfer feature. Sometimes applications need to transfer larger payload than the payload that fits into single message. This type of data slicing is handled well at application level than lower level (e.g. IP layer or adaptation layer). In CoAP, Block1 and Block2 options are provided for transferring request payload and response payload respectively in block-wise fashion. The lightweight IoT platform is built using Node.js. We implement Block1 feature for node-coap library which is CoAP implementation for Node.js.**

***Keywords*—** CoAP, Internet of things, Node. js, Visualization, Virtualization.

I. INTRODUCTION

The IoT devices work in constrained environment i.e. low memory and power. Hence, the platform should be lightweight and since these devices need to interact with each other on Internet, the communication protocol should also be lightweight. Most IoT platforms are heavy-weight and they require proprietary protocols. The lightweight IoT platform uses node.js and mongoDB which are open source software [1]. This work can be enhanced with CoAP, which is low energy protocol for M2M environment [5].

There is limit on the maximum data size that can be sent in the CoAP message. CoAP works well for small messages e.g. sensor outputs. Sometimes applications require to send large data through request or response. CoAP is based on UDP and UDP needs IP fragmentation to send data larger than MTU. This kind of fragmentation lowers the performance of constrained networks. The data slicing can be managed better at application layer than lower layers. So in the CoAP, block-wise transfer feature is provided. Payload can be sent with both request and response. For request payload block1 option is used and for response payload block2 option is provided [6].

We implemented lightweight IoT platform using CoAP focusing on main features of IoT platform which are virtualization, visualization, database and REST API. This platform is made up of Node.js which is server side javascript technology and MongoDB which is a No SQL type databse [1].

II. SYSTEM ARCHITECTURE

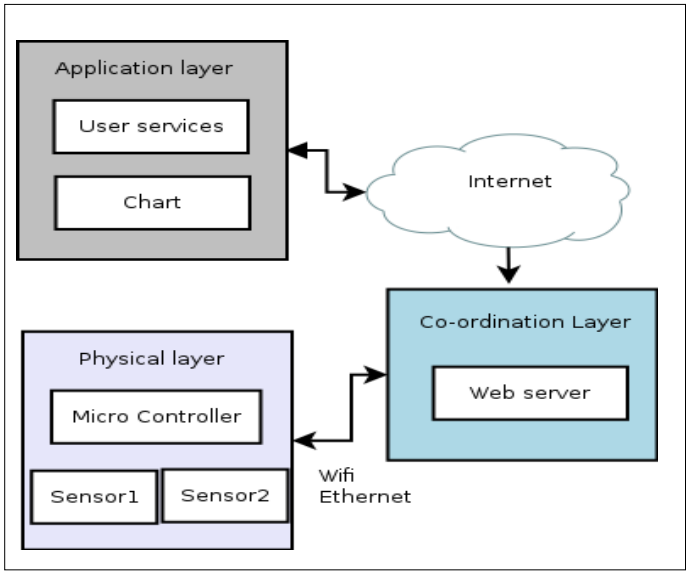
Our system architecture is comprised of three layers.

Hardware layer : This layer contains sensors and actuators. These IoT devices communicate with web server through Ethernet or wifi. These devices send their data to server. We will use RaspberryPi and DHT11 sensors. RaspberryPi is a single-board computer and DHT11 are sensors for measuring temperature and humidity.

Co-Ordination Layer : This layer contains web server which processes data coming from IoT nodes and stores data in the database. The nodes use CoAP for communication. We use Node.js to build web-server. Through web page user can access IoT devices and virtualize them. Data coming from sensors is stored in MongoDB database. The IoT devices communicate using RESTful API. Like HTTP, CoAP also supports REST architecture. Sensors devices send their status data to the web server through http/coap post request. Web server stores this data in the MongoDB database.

Application Layer : This layer is comprised of applications and users of systems. Through Internet user can access devices and control them. e.g. User can on/off bulbs in the home through web page. Also user can create chart using the database kept for history.

Figure 1 System Architecture.



II. METHOD

Figure 2 explains algorithm for CoAP block1 transfer.

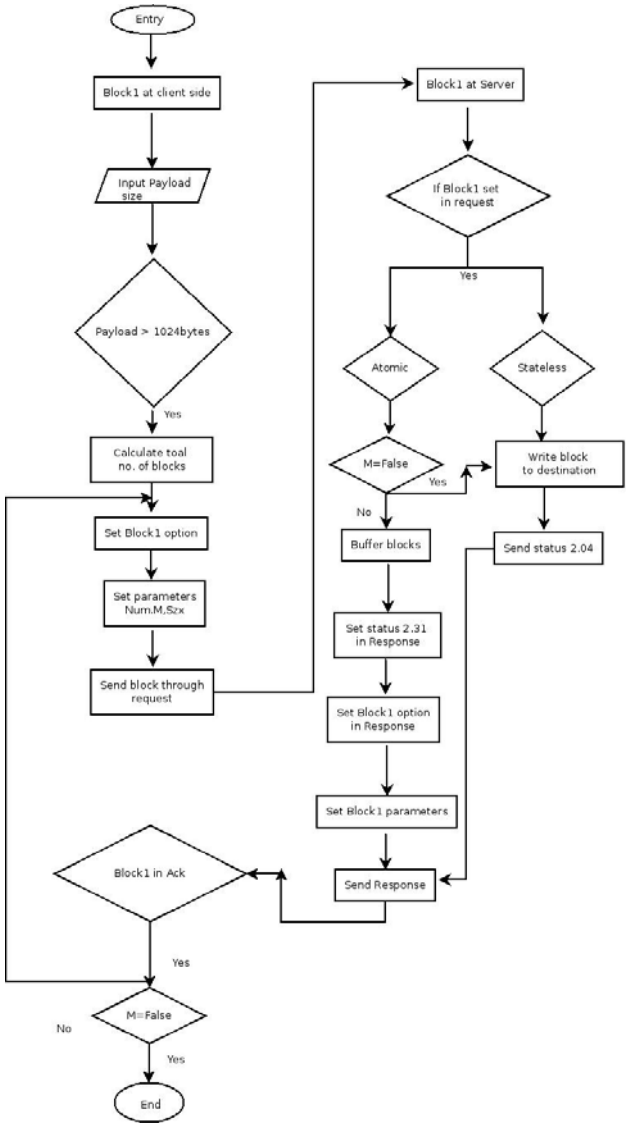


Figure 2. Algorithm for CoAP Block1 transfer

III. IMPLEMENTATION

Hardware components : Actual lightweight platform implementation setup will be done with RaspberryPi and DHT11 sensors. Currently we are simulating the sensor results.

Software components : Operating system - debian GNU/Linux,

Node.js - version v0.10.29,

MongoDB - version 2.4.10.

A. *Virtualization and Server using node.js*

Virtualized icons of IoT devices are displed on web page. Through web page user can access devices. e.g. User can on/off bulb using the touching the icon on web page.

```
var app = require('../app');
var http = require('http');
var port = normalizePort(process.env.PORT || '80');
var server = http.createServer(app);
```

Figure 3 : Running HTTP server

```
var coap = require ('coap');
var server = coap.createServer(function(req, res)
server.listen()
```

Figure 4 Running CoAP server

Node.js is server side scripting technology which is built on google's V8 engine. Node.js is event-driven, single threaded and asynchronous framework. It is designed to perform fast and focusing on low memory consumption. Figure 3 and Figure 4 show how to run simple web server using http and coap respectively.

The core logic of starting HTTP based server on default port (port 80) can be seen above.

app :- Its a global app configuration and is used from express middle ware.

http:- Its a module (npm install http) which is used to support

HTTP protocol.

B. With CoAP, to start a similar server but using CoAP protocol instead of HTTP, following code is used:

coap :- It's a module(npm install coap) which is used to support coap.

B. *Database using MongoDB*

MongoDB is referred as NoSQL database as it does not use traditional relational model of SQL. It is open source and cross platform database developed for fast execution of queries. Figure 5 shows how to insert data in the database using mongodb.

```
Var db = require ('mongodb').MongoClient
MongoClient.connect(url , function (err, db){
        if(err)
                console.log(err);
                else{
                        data = JSON.parse(req.payload);
                        console.log("Successfully connected to
the database");

db.collection('Temperature').insert({"Value":data.Value ,

“Time:data.Time”})
                        db.close();
                }
})
```

Figure 5 : Inserting sensor data to the MongoDB database

mongodb : Node.js module for MongoDB
MongoDb provides simple API for accessing database (create, insert, update, delete).As shown in figure 4. data coming from temperature sensor is stored in the table(collection) Temperature. We can easily access the data as shown in Fig 6.

```
namrata@debian:~/working/IoT$ mongo
MongoDB shell version: 2.4.10
connecting to: test
> db.Temperature.find()
{ "Value" : 19, "Time" : 1453879038967, "_id" :
ObjectId("56a86efff8233e89692ffb54") }
{ "Value" : 21, "Time" : 1453879038960, "_id" :
ObjectId("56a86efff8233e89692ffb55") }
```

Figure 6 : View database

C. *RESTful API*
As shown in the figure 7. CoAP post method is used to transfer data between devices. CoAP POST request is designed to send enclosed data within body to the web server.

D. Visualization
To implement visualization we use Plotly. Plotly provides Node.js library for making charts. It provides a way to visualize a data on web site. We can put MongoDB data into chart using Plotly.

E. CoAP Block-wise Transfer
We implemented Block1 feature for node-coap library. Block1 option is necessary when there is a payload larger than 1024 bytes to be sent through request. This option can be used with POST and PUT method as payload within the request can be transferred using these two methods only.

```
var options = {
   host : "localhost",
   pathname : "/insert",
   method : "POST"
   headers: {
        'Content-Type': 'application/x-www-form-
                        urlencoded',
        'Content-Length': Buffer.byteLength(data),
     }
        };
var req = coap.request(options);
req.write(JSON.stringify(data));
```

Figure 7 :CoAP POST request

1)Structure of block option
When request payload is larger than 1024 bytes the 'Block1' option is enabled in the request.The block1 option carries three fields of information- Num, M, SZX.
Num - Number of block
M - Whether more blocks are following or not
SZX - Size of block in bytes
Block option value is 0 to 3 bytes unsigned integer. Its first three least significant bits gives size of block which is encoded value from 0 to 6 (i.e. 0 for 2^ 4 bytes to 6 for 2 ^ 10 bytes). Actual block size is calculated as 2 ^ (szx+4). The M bit (4th least significant bit) indicates whether there are more blocks or not and Num field is obtained by dividing block value by 16 [6].

2) Usage
Descriptive usage (Block1 option in request)-
Num - Number of block which is provided ,
M (more bit flag) - when unset indicates that there are no more blocks to transfer and
SZX - size of the block being provided.

Control usage of block1 (Block1 option in acknowledgement)-
In the atomic fashion blocks are buffered at the server and written to the destination when all the blocks received.. In stateless fashion block is written to destination immediately when received, it is not buffered.
We implemented both atomic and stateless server.
Num- Number of block which is being acknowledged,
M - if set then indicates that body of message is buffered (atomic fashion) and if unset indicates that block is written to the destination (stateless fashion).
SZX- size of block being acknowledged.

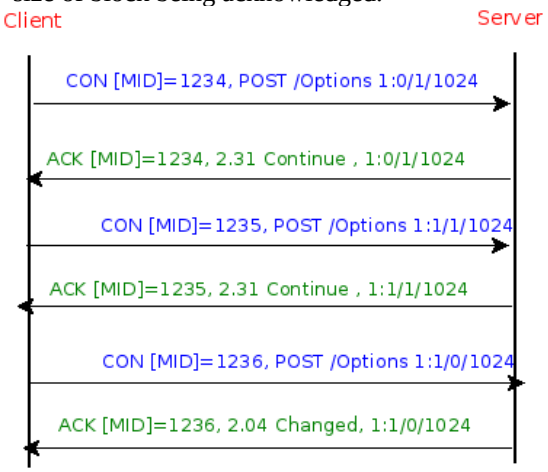


Figure 8 Simple Atomic POST Request

Figure 8 shows the simple atomic block-wise POST request. The meaning of atomic is that the provided resource representation is changed at the server side when all the payload blocks will arrive. CON is Confirmable message, MID is message ID, ACK is acknowledgement, 2.31 status tells that message body is buffered at server, 2.04 status tells
that resource has been changed with all blocks of data. In case of failure server should return the error code '4.08'.[6]

IV. RESULTS

Web page containing virtualized icons of devices is as shown in the Figure 9.
We implemented http client-server using node.js http library and CoAP client-server using node-coap library. We compared http packet size and coap packet size using Wireshark. The
Results are shown in the Figure 10 and 11.

Figure 9. Vitualized Icons of IoT devices

As we can see in the figures 10 and 11 HTTP requires 201 bytes to post some data from client to server and CoAP requires 72 bytes to post same data.

We implemented block1 feature and sent image of size 3.4kB through CoAP POST request. As the image is larger than 1024
bytes, block1 option is enabled and image is sent using 4 blocks of 1024 bytes. The Wireshark statistics for block1 option with simple atomic POST request is shown in figure 12.

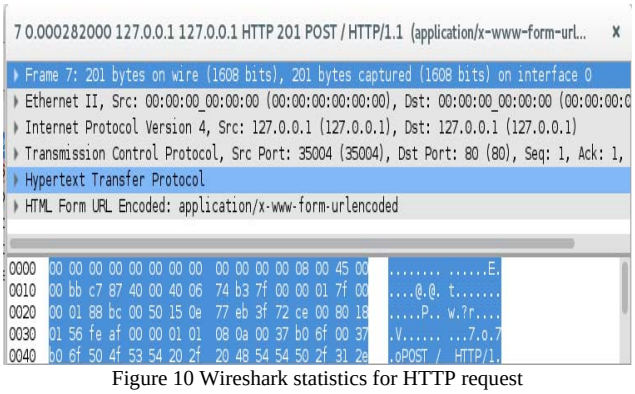


Figure 10 Wireshark statistics for HTTP request

The Wireshark statistics for block1 option with stateless POST request is shown in figure 13.

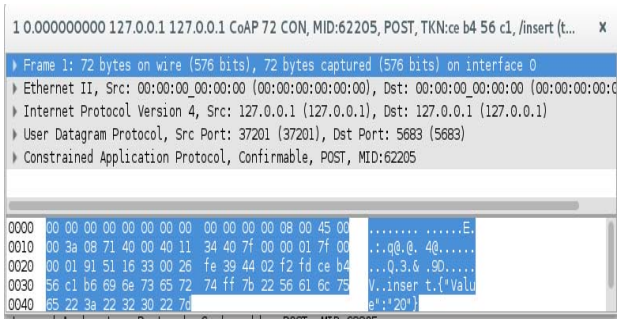


Figure 11 Wireshark statistics for CoAP request

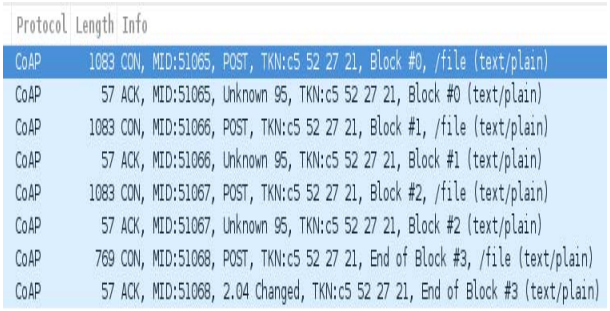


Figure 12 Atomic Block1 Transfer

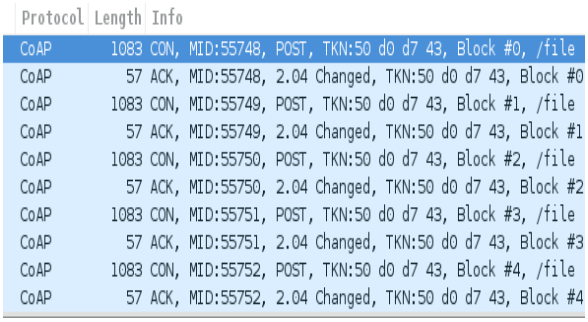


Figure 13 Stateless Block1 Transfer

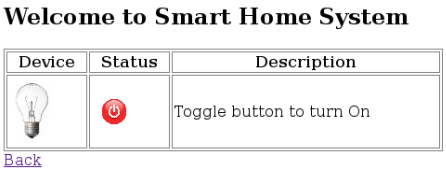


Figure 14 Accessing devices from web page

V. CONCLUSIONS

In this paper, we presented lightweight IoT platform with support for CoAP Block1 transfer, developed by node.js using
constrained application protocol as an alternative to conventional HTTP method. With the use of new improved lightweight IoT platform, device's energy can saved by sending the same information with much smaller packet overhead. Our experiments suggest that 65% of overhead reduction can be achieved using CoAP with lightweight IoT platform.
We optimized the framework for large size payload. e.g., IoT device which periodically sends captured images to web server. Thus, we provided block1 support for Node.js.
We can further optimize the system to use 6LoWPAN which is an acronym of IP6 over Low power Wireless Personal Area Networks. Also we can add DTLS security to the platform.

ACKNOWLEDGMENT

I would like to thank ME Computer department, PICT.

REFERENCES

Conference References

[1]. Young Ju Heo, Sung Min Oh, Won Sang Chin, Ju Wook Jang, "A Lightweight Platform Implementation for Internet of Things",3rd International Conference on Future Internet of Things and Cloud 2015.

[2] J. Mineraud, O. Mazhelis, X. Su, and S. Tarkoma, "Contemporary Internet-of-Things platforms", http://arxiv.org/abs/1501.07438, Jan 2015, technical report.

[3] Gubbi, J.Buyya, R.Marusic, S.Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions" Future Generation Computer Systems, 29(7), 1645-1660.

[4] Kovatsch, Matthias, Simon Duquennoy, and Adam Dunkels, "A low-power CoAP for Contiki",Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference on. IEEE, 2011.

Website References

[5] https://tools.ietf.org/html/rfc7252

[6] https://datatracker.ietf.org/doc/draft-ietf-core-block/

[7] https://github.com/mcollina/node-coap

[8] www.tutorialspoint.com/nodejs/

[9] docs.mongodb.org/manual/reference/sql-comparison

[10] https://plot.ly/nodejs/

Book References

[11] Cantelon, Mike, et al. Node. js in Action. Manning, 2014.