Device Microagent for IoT Home Gateway

A Lightweight Plug-n-Play Architecture

Dhiman Chattopadhyay
Tata Consultancy Services
Kolkata, West Bengal, India
dhiman.chattopadhyay@tcs.com

Abinash Samantaray
Tata Consultancy Services
Hyderabad, Telengana, India
abinash.samantaray@tcs.com

Anupam Datta
Tata Consultancy Services
Kolkata, West Bengal, India
datta.anupam@tcs.com

ABSTRACT

Smart home implementation in IoT involves practical challenges of management and scalability of connecting various non IP end-devices i.e. sensors and actuators behind the connnected home gateway.While there are separate standards for interaction between IoT service to home gateway and gateway to variety of end-devices there remains disconnect regarding how this two ends meet in an adaptable and scalable way. In this paper we present an light-weight,loosely coupled architecture for IoT smart home gateway whereby end-devices can be added dynamically on the gateway without disrupting long haul communication between IoT cloud service and gateway.The gateway agent exchanges data through sensor-block or actuator-block with end-devices via device microagents and the protocol specific read-write task is offloaded to individual device microagent. This hybrid approach to integrate MQTT pub/sub flexibility with LWM2M RESTful adaptability results in a design of plug-n-play modular agent architecture for home gateway management in IoT applications.

CCS CONCEPTS

• **Computer systems organization → Embedded systems**; **Sensor networks**; **Sensors and actuators**;

KEYWORDS

Microagent, Sensor-block, Actuator-block, LWM2M, MQTT, JSON, TCUP

ACM Reference Format:
Dhiman Chattopadhyay, Abinash Samantaray, and Anupam Datta. 2017. Device Microagent for IoT Home Gateway. In *Proceedings of Advances in IoT Architecture and Systems, Toronto, Canada, June 2017 (AIoTAS'17),* 8 pages.

1 INTRODUCTION

Internet of Things (IoT) bridges the physical world with the digital world by connecting various devices to facilitate data collection,analysis and actuation for specific action [4]. Smart home is an usecase in IoT where home devices are connected to the internet and remote users can get device status and send command to devices. Inside a smart home various end-devices like temperature controller,energy meter,smart bulbs are connected to a home gateway which in turn communicates with the backend IoT service hosted in cloud. The end-devices,sensors or actuators, interact with home gateway over various short-haul protocols like Zigbee, Z-Wave, BACnet, Modbus, UPnP, Serial [13] ; and the gateway is connected to the IoT cloud over a TCP/IP based long-haul protocol. Device modelling plays an important role in cyber physical system (CPS) to capture device or sensor characteristics into software mapping,here semantics and constraints are also captured. This research [5] describes a comprehensive seven layer model of sensor for interoperability and one usecase on such comprehensive sensor modelling for smart meters is explained in this paper [7]. However this sensor model does not address the modeling of multiple heterogeneous sensors behind the home gateway. Device Management(DM) is the mechanism through which an authorised user can remotely set device parameters, query device details, capture device data, conduct troubleshooting, install or upgrade software in devices. For smart home scenario DM is required to manage the home gateway as well as the end-devices behind the gateway.

AIoTAS'17, June 2017, Toronto, Canada

1.1 Evolution of DM Protocols

Initially device manufacturers developed propreitary mechanism for communication and management of their devices. Gradually standards evolved to ensure uniformity and interoperability and some of these gained traction in industry.SNMP emerged as standard for management of network equipments. Then telcos and broadband service providers introduced a protocol TR69 for management of broadband routers and cable modems. Next OMA DM emerged as the device management standard for mobile devices. To cope with the constraints of M2M device Open Mobile Alliance (OMA) introduced LightweightM2M and major industry players are backing it. MQTT is another lightweight protocol emerging as a popular protocol in IoT. While OMA has already come up with LightweightM2M standard some useful features (like gateway management object) of OMA DM standard were left out as redundant. However that resulted in some limitation in gateway management. So there is a need for scalable lightweight DM agent for gateway to interface heterogeneous sensors data in a protocol agnostic manner. Let us look at major DM standards to understand the evolution of device management and identify the implementation challenge from gateway perspective.

1.1.1 SNMP. Internet Engineering Task Force(IETF) has defined Simple Network Management Protocol(SNMP) for managing IP connected devices like routers,printers,switches etc.SNMP is an application layer protocol that operates over IP or UDP defining set of interfaces for network management,a data model and a set of data objects.SNMP exposes management data in device side agent in the form of objects to describe the system configuration. Device agent talks with Network Management Server(NMS). The device can be queried and updated by managing applications.The device data model is called Management Information Base(MIB) that forms hierarchic namespace consisting of Object Identifier(OID) and each OID is associated with a device parameter.Operation supported are get ,set and trap notification.

1.1.2 TR69. Broadband Forum's TR69 standard is another application layer protocol based SOAP over HTTP. Communication between Customer Pemise Equipment(CPE) and the server Auto Configuration Server(ACS) based on CPE WAN Management Protocol(CWMP). Different phases in TR69 device lifecycle are provisioning, authentication, service activation, software update and remote monitoring. This protocol supports devices behind router through NAT using STUN protocol.The standard defines the data model and operation interfaces

1.1.3 OMA DM. OMA Device Management(OMA DM) protocol is based on client server architecture where devices act as client and communicates with DM server that interface between users and devices. The device agent comprises of device resources like static or dynamic device parameters, configurations, actions, alerts etc. OMA DM models the device as a set of resources in the form of a hierarchial tree and exposes RESTful management objects to the DM Server. DM client is made up of different resource groups called Management Objects (MO) facilitating specifics for the management of devices.OMA DM defines data model of various management Object like Device, Firmware, Location, Server, Access Control etc. OMA DM uses HTTP as transport and uses SyncML format for data exchange OMA DM Gateway Management Object(GwMO) [1] provides a mechanism for OMA DM framework to manage end-devices/sensors connected to a gateway where direct connections between server and end-devices do not exist due to non existence of publicly routable address for devices as devices may be placed behind a firewall.Three types of management modes are supported : (a)Transparent Mode where the DM Gateway assists the DM Server in forwarding a server message to the end-devices connected behind gateway (b)Proxy Mode where the DM Gateway,on behalf of the DM Server, acts as intermediate manager for end-devices over OMA DM protocol (c)Adaptation Mode where the DM Gateway,on behalf of the DM Server,manages end-devices over non OMA DM protocols.

1.1.4 OMA LWM2M. OMA introduced Light-weight Machine-to-Machine(LWM2M) standard which follows a client-server architecture [2]. LWM2M uses the Constrained Application Protocol(CoAP) over UDP [12] as underlying trasnport protocol. CoAP follows a HTTP like RESTful architecture and similar status codes but consumes less resources than HTTP.Thus CoAP is a better choice for constrained M2M devices that are less resourceful in terms of CPU, memory footprint, power and bandwidth[17]. LWM2M has also standardized CoAP-HTTP proxy as a bidirectional translator between two protocols. LWM2M client i.e. device agents communicate with DM server over CoAP and DM service exposes HTTP RESTful API for device data consumption or sending back commands to device. LWM2M client hosts a tiny CoAP server and each property of a device is modeled as a CoAP resource and similar set of features are grouped under a LWM2M management objects. CRUD operations can be performed on the management objects and resources of device. Eight normative objects are defined in LWM2M which are LWM2M Server Access Security, LWM2M Server, Access Control, Device, Connectivity, Firmware, Location and Connectivity Statistics. However developers can add their custom object and resources. LWM2M supports DTLS as transport layer security.

1.1.5 MQTT. Message Queue Telemetry Transport (MQTT) protocol is an ISO standard and MQTT v3.1.1 is part of OASIS specification. MQTT works over TCP and resembles a hub and spoke model where every participant, devices or applications,are connected with a message broker [14].MQTT is a lightweight protocol ideal for IoT due to lower consumption of battery and bandwidth than HTTP [20]. Any participant can publish to the message broker with specific topic and similarly can subscribe to any topic as well. As soon as a message gets published into the broker all subscribers, listening on that topic, will automatically get that message. Moreover MQTT allows topic hierarchy and allows multilevel(#) or single level(+) wildcards for subscription for multi-pattern or single pattern of topics. Published events can be durable or nondurable while QoS for MQTT can be defined at three level - fire and forget,at least once(default) and exactly once. MQTT-SN is a variation for MQTT for non IP embedded devices or sensors. There are implementatio to bridge both MQTT with REST for IoT [9].In IoT MQTT is used as the underlying backbone for targeted data exchange where devices can publish their data with defined topic and the application listening for those device topics will automatically get the data. Similarly any command intended for the device can be published to that device from application end via same pub-sub mechanism. Kim et al. presented a MQTT based device management [15]. MQTT brokers has configuration option for password based authentication through TLS/SSL to ensure security. However there is a limitation in maximum number of topics a single MQTT broker can support; so one MQTT broker may soon run out of topics in a city wide IoT deployment thus needs clustering to scale up.

As the IoT is becoming a protocol jungle there will be different school of thoughts on choice of protocols. However each is having its own pros and cons thus "one size fits all" approach does not hold true for all IoT use-cases. Hence OneM2M partnership came up with an inclusive approach to form alliance among different standardization bodies for coordinating and avoiding duplication of concepts. As IoT developer and system integrator we need to be judicious for optimal adoption of the standard(s) based on scalability and flexibility.

2 TCS CONNECTED UNIVERSE PLATFORM OVERVIEW

TCS Connected Universe Platform(TCUP) is a Platform as a service (PaaS) offering for IoT solutions across verticals [19]. TCUP provides a multitenant platform offering essential IoT services like Sensor Data Management, Device Management, Message Routing, Complex Event Processing, Task, Analytics and Visualisation modules as loosely coupled building blocks for developing IoT solutions. TCUP services can be hosted on either in-premise private cloud (like Openstack) or commercial public cloud (like AWS or Azure).Each of the TCUP services exposes their RESTful API. The authentication and authorisation for TCUP access is controlled by tenant key. All RESTful API calls pass through the API gateway that does authentication, authorization and rate control. Figure 1 shows TCUP high level architecture. TCUP is used in IoT applications across various

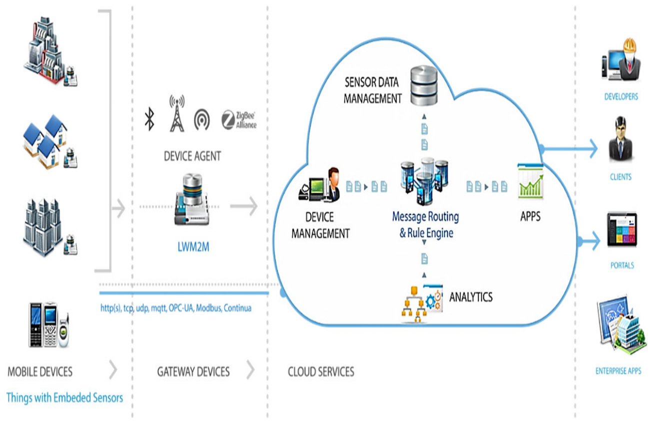

Figure 1: TCS Connected Universe Platform.

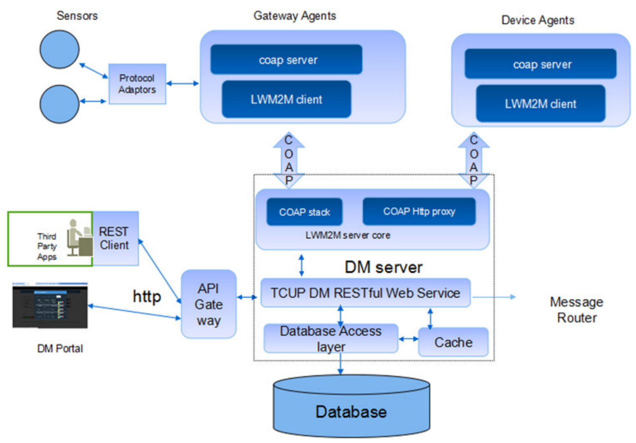

Figure 2: TCUP DM Service highlevel overview.

industry verticals like campus energy consumption monitoring,bus fleet monitoring, smart window shade control,smart mediacentre control,remote device test automation,surveillance solution [6] etc.

2.1 TCUP Device Management

TCUP DM follows OMA LWM2M standard based client server architecture where the device runs the LWM2M client and the DM server hosts the LWM2M server. Figure 2 shows Device Management Service(DMS) high level architecture. CoAP stack with COAP HTTP proxy forms the core LWM2M enabler of the DM server.Rest layer and data persistence layer sits on top of core LWM2M stack and LWM2M server with a REST layer forms DMS. PostgresSQL RDBMS is used for data persistence. A DM user is a registered device owner and a member of an active tenant of TCUP platform. Every device user is given a user credential and a DM user key plus TCUP tenant key,these need to be passed on with every DM RESTful API call for user authentication.Device portal is a web application consuming DM RESTful services. The DM user can view on device portal all devices registered against the user key. DM agents are run on edge devices to register those with the TCUP DMS. The events from any device observable resources ,like sensor data, are posted to the DM. DM forwards the traffic to Sensor Observation Service module via Message Router service which acts as a broker. This makes TCUP modules loosely coupled to ease integration of device data with any third party application.One of the challenge in LWM2M implementation is to keep the NAT alive to access devices behind network routers,we achieve it by sending periodic heartbit from device. There is an optional OTA update microservice component that facilitates over-the-air update of device software. That requires running an OTA daemon on device that can intercept any update request and invoke callback function to download patch,check integrity of downloaded patch,install and reboot. Obviously the implementation varies based on platform and operating system. Following processes define major device life cycle phases in TCUP DM as per OMA LWM2M.

2.1.1 Bootstrap and Registration. Bootstrap operation is used to provision DM agent with the bootstrap information i.e. DM server access details (like URI, access security object) in order to register

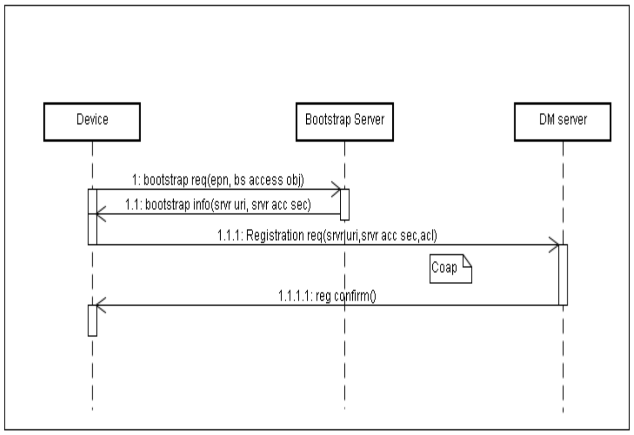
Device
Bootstrap Server
DM server
1: bootstrap req(epn, bs access obj)
1.1: bootstrap info(srvr uri, srvr acc sec)
1.1.1: Registration req(srvr uri,srvr acc sec,acl)
Coap
1.1.1.1: reg confirm()

Figure 3: Sequence diagram for Bootstrap and Registration.

with the DM Server.Bootstrap mode can either be factory burn of bootstrap information into device for client initiated bootstrap or server discovery boostrap.In TCUP DM we follow client initiated booototrap where new device initially connects to a bootstrap server to get bootstrap information comprising of Server account information and Server access security object for registering with DM server.

Registration process commissions a new device with the DM server,which is notified about the new device ID, IP and its resources along with access control list (ACL) which defines permissible operations on device resources. The protocol also ensures automatic registration update known as reregistration for sending periodic update of device IP and resources to DM server. Bootstrap and registration process is shown in sequence diagram 3.

2.1.2 Device Query and Update. This process supports configuration setting,parameter reading and connectivity status query. The query may be group level query or individual query on devices. This process also facilitates dispatching and executing user commands like reboot,lock device through DM service call.Developer needs to

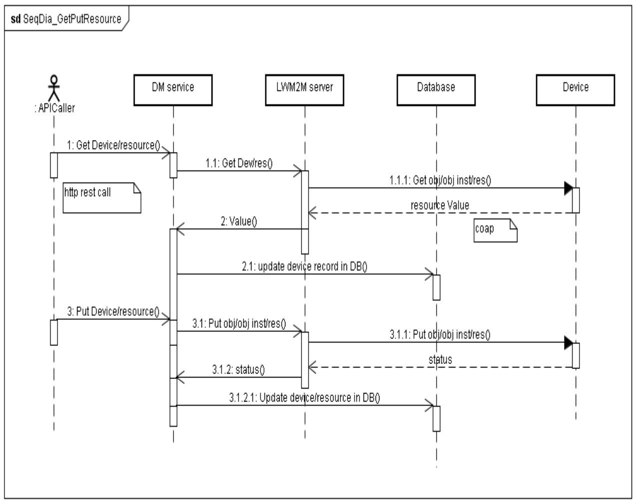

Figure 4: Sequence diagram for Query and Update of resources.

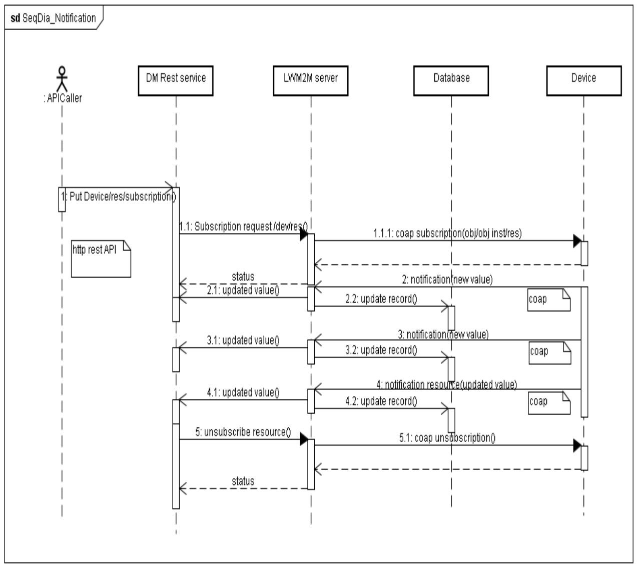
sd SeqDia_Notification
: APICaller
DM Rest service
LWM2M server
Database
Device
1: Put Device/res/subscription()
1.1: Subscription request /dev/res()
1.1.1: coap subscription(obj/obj inst/res)
http rest API
status
2: notification(new value)
2.1: updated value()
2.2: update record()
coap
3: notification(new value)
3.1: updated value()
3.2: update record()
coap
4: notification resource(updated value)
4.1: updated value()
4.2: update record()
coap
5: unsubscribe resource()
5.1: coap unsubscription()
status

Figure 5: Sequence Diagram for Subscription Notification.

write callback function to implement actual operation triggered by command. Following sequence diagram 4 shows device read and write process.

2.1.3 Observe Notification. The process enables event subscription mechanism to opt-in and opt-out for notifications from observable resources. Automatic notifications will occur from device to DM server either on a value change or on satisfying a condition like crossing a threshold. Further the events may be forwarded to a message queue to be processed by other applications. Following sequence diagram 5 shows observe notification process.

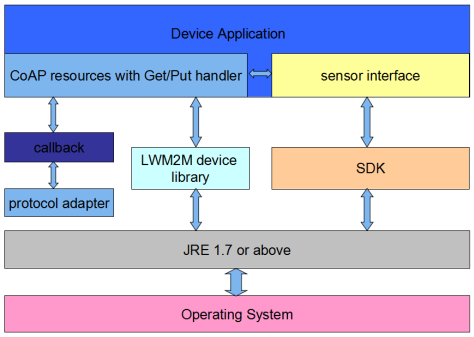

Figure 6: TCUP DM agent highlevel block diagram.

2.2 TCUP DM device agent

A device runs DM agent that registers with DM server. DM agent is basically a LWM2M client that hosts a CoAP restful resources.Every device parameter can be modeled as a CoAP resource. So resources may be static (like IMEI etc.) or dynamic (like free RAM, sensor reading etc.), certains resources (like address) may be editable through rest interface based on the ACL permission. Object is a logical grouping of similar type of resources - for e.g. location object comprises of latitude, longitude and altitude resources.Any DM resource can be made observable for automatic notification of the resource value. A resource serves its read write request through its GET/PUT handlers. Every resource serves query or update service call by its GET or PUT handlers respectively. Once a device is registered users can log into DM portal to view devices;all device resources are listed under different tabs, namely Device, Hardware, Software, Location, Sensor and Custom. User can read device parameters, update editable parameters, send commands to devices and subscribe for automatic notification from dynamic resources like sensors. These functions are executed through DM RESTful APIs which are consumed by the DM portal and any other application consuming device data. Figure 6 depicts TCUP DM agent high level block diagram. However the limitation with present DM agent architecture is that customisation is required with any addition of new end-devices as corresponding new resource needs to read from different sensors over their native protocols.However plug-n-play implementation exists in other protocols like UPnP to allow discovery and joining of new entrant in peer to peer communication. Even LWM2M predecessor OMA-DM standard supports GwMO concept where OMA DM server maintains hierarchical tree of gateway management objects for end-devices connected behind the gateway and GwMO interacts with devices for management; any change is handled through change in GwMO adapters and the protocol supports dynamic alteration of those GWMO. However LWM2M lacks GwMO support. There is a scope of improving the LWM2M gateway agent for connecting end-devices in a lightweight but protocol agnostic manner.Next section describes our proposed architecture where plug-n-play flexibility is achieved for run-time onboarding of end-devices.

3 PROPOSED ARCHITECTURE - DEVICE MICROAGENT

Our proposed architecture tries to bring in plug-n-play flexibility for dynamic onboarding of end-devices into LWM2M gateway agent.Resource constraints in IoT forced our choice to lightweight protocols. XMPP is ruled out because of its similar overheads like HTTP. Sheltami et al. described how publish subscribe model is successfully used in wireless sensor networks [23]. MQTT is more lightweight than AMQP or STOMP because it draws less power and consumes less bandwidth [26]. As the proposed gateway agent architecture is designed by adding a message broker,MQTT is chosen as the pub-sub backbone beneath a LWM2M stack that offers REST interface over CoAP ,another lightweight protocol.Our earlier work [8] proves that CoAP consumes only 10% of bandwidth of HTTP to transmit same data. To avoid tight coupling between device resource and end-device a loosely coupled agent architecture is presented where a gateway reads or writes through a sensor-block or actuator-block to group of end-devices via microagents. A sensor-block is formed by aggregated list of sensors,similarly an actuator-block is aggregation of actuators. The gateway LWM2M agent hosts CoAP resource for respective sensor-block or actuator-block that will communicate only through an uniform payload structure using a data serialization format like XML, JSON, EXI, CBOR, BSON etc. The sensor-block payload is produced or consumed by individual end-device MQTT clients called DM microagent.The sensor-block payload can be published or subscribed through the microagents connected via MQTT broker.Thus there is no direct sensor interaction from gateway resource's GET/PUT handlers. Figure 7 shows highlevel architecture of proposed gateway agent with microagents. Microagents read the individual sensor data and publish on defined topics which are subscribed by sensor-block resource and vice versa. MQTT plays the role of intermediate message broker between sensor-block resource and end-device microagent adapters. The routing of published or subscribed events on MQTT is governed by topic ; so new MQTT topics for additional end-devices can be fed into the gateway agent though DM service to tell agent to listen for new topics. Every query or update call on sensor-block or actuator-block resources reaches GET/PUT handler of respective resource. For reading sensor value the sensor-block resource GET handler subscribes to topic *reading/sensor/#* on the MQTT broker,where sensor data is published on topic *reading/sensor/id* by different microagents actually reading the sensor values. For sending command to device the actuator-block resource PUT handler publishes command payload on topic *control/type/id* to MQTT broker. Corresponding actuator microagent subscribes to topic *control/+/id* to get the command and execute. Sensor-block or actuator-block resource may serve multiple microagents by treating set of sensors as one composite resource; communication with each sensor is achieved by parsing and iterating through the composite payload schema of sensor-block . The composite payload is extensible as the sensor-block data will grow further with inputs from additional microagent. The topics can vary based on the usecase. The IoT application can read sensor-block or write actuator-block payload through DM service,the parsing of individual values or constructing the composite payload is responsibility of IoT application. Customization is not required in LWM2M gateway agent

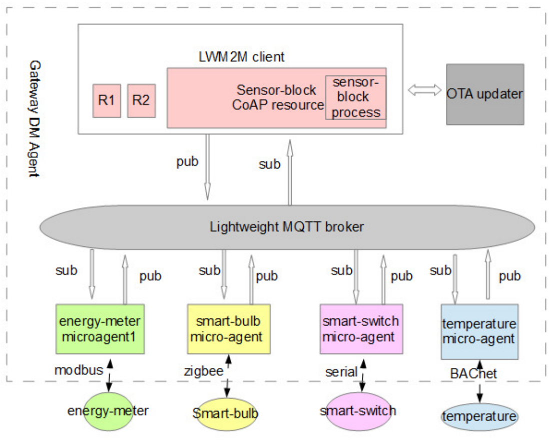

Figure 7: TCUP DM Microagent architecture.

while adding a new microagent suffice to support a new sensor or actuator i.e. end-device. Additional microagents can be deployed through TCUP OTA update module. The advantage is decoupling of individual sensors or actuators access from gateway LWM2M agent as the job of interfacing and processing sensor values is offloaded to microagents. Generally sensor event lister libraries are event driven in nature the microagents also work in event driven manner.

Device management in other popular IoT Platforms are either based on MQTT (like Amazon IoT,Azure IoT,IBM Watson) or CoAP (like ARM mbed IoT ,Siera Wireless Airvantage).While MQTT offer pub/sub flexibility CoAP is ideal for command triggering. MQTT performs better in latency while CoAP excels in bandwidth requirement[25]. Our architecture is unique as we try to leverage the best of both lightweight protocols.Here device to cloud communication is over CoAP unlike the ESR proposal of LWM2M over MQTT [10] as we want to retain request/response RESTful architecture along with pub/sub flexibility.

However addition of MQTT broker increases load on gateway computation resource, but considering the availablity of memory and CPU power in modern rugged gateway like Eurotech ReliaGATE this is not a bottleneck. Practically the number of end-devices in a smart home is not large, so MQTT broker clustering is not required in home gateway.

A sample sensor-block json payload is listed below :

```
{
"sensor-block": [{
   "topic": "reading/temperature/tem101",
   "protocol": "zigbee",
   "unit": "cel",
   "ts": 1491462377641,
   "value": 24.3,
   "datatype": "float"
  }
 ]
}
```

A sample actuator-block schema is shown below :

```
{
```

```
"actuator-block": [{
   "topic": "control/light/lt103",
   "protocol": "zigbee",
   "ts": 1491462378627,
   "command": {
   "instruction": "off",
   "param" : "nil" }
  }
 ]
}
```

Payload schema may extend and evolve to be more generic structure encompassing various sensor parameters based on type of sensors or actuator used.Intelligent design of this uniform payload schema will minimise the development effort at the microagent level.

3.1 Choice of Technology

Choice of MQTT broker is decided based on three criteria lightweightness,performance and openness.Two popular open source MQTT broker Mosquitto and Emqqtd are written in C and Erlang respectively. This study [3] shows that C based Mosquitto is the fastest among peers. A stress test study of Mosquitto [27] shows it can handle at least 20k concurrent connections at a speed of 7000+ events per second consuming just 12MB of RAM on a single core 2.1GHz 4GB virtual Ubuntu server on VMWare. There is an upper limit of number of topics that can be supported by one Mosquitto instance(typically over 100K),however there will not be scaricity of microagent MQTT topics in a home gateway as number of end-devices in a smart home can not run into thousands.There is no question of network lattency between MQTT broker and LWM2M agent as both runs on same host i.e. gateway. Choice of data serialization format is a debated subject in IoT [24]. ETSI M2M prescribed format JSON is adopted in our first implementation as the most popular format.Selection of JSON parsing library is based on performance benchmark survey [22] where Jackson,a popular Java JSON library,emerged as a top performer; so we selected Jackson in our microagent software stack.

3.2 Performance Estimation Model

There is not many performance prediction model in IoT DM however a research is done on sensor data storage[11].Our proposed architecture has two major layers - LWM2M stack working on request/response principle and MQTT based microagents working on pub/sub fashion. S. Oh et al. published an mathematical model [21] to estimate performance impact for pub/sub vis-a-vis request/response model.Assuming event generation and consumption follows Poisson process pub/sub round trip time (RTT) approximately equals the pub-sub cost per event generated.We assume same cost analysis holds true in our MQTT layer. For the LWM2M part a performance estimation model may be conceptualized as a hypothesis describing the factors that impact the response times. We apply queueing theory to estimate a relationship between the response time for LWM2M requests and the number of concurrent Rest requests (GET/PUT).Assuming a steady state queue Little's law can be applied in the performance measurement as per the following formula : [18]

$$N = TP * (RT + TT), \tag{1}$$

where N is the number of concurrent request, TP is the throughput, RT is the response time, and TT is the think time.

TP is dependent on computing capacity of the home gateway and load on the gateway. The relationship between response time and the number of clients may be assumed to be a non-linear function like a simple M/M/1 open queuing model:

$$RT = ST + QT = ST + (ST * u/(1 - u)), \tag{2}$$

where RT, ST, and QT are the Response Time (RT), Service Time(ST) and Queue Time (QT), respectively, and u is the utilization of bottleneck computing resource that serves the web request . The response time increases exponentially due to lack of resource as the utilization of this resource approaches its full capacity limit. Kingman's formula can be used to estimate QT [16]. Now in our proposed architecture ST is also dependent on latency due to message publishing and subscribing by microagents plus the sensor-block payload processing overhead k which will be constant for a given payload schema.So time delay between event occurrence and notification to subscriber equals to (p+s) where p is latency for publish and s is latency for subscribe of sensor events.If we assume probability of publishing of any event e is P(e) then modified service time can be estimated as :

$$ST' = ST + (p + s + k) * P(e). \tag{3}$$

We can assume event publishing is a Poisson process to calculate P.So that results modified response time as :

$$RT' = ST' + (ST' * u/(1 - u)). \tag{4}$$

Obviously this model holds true on assumption of a steady state system with all microagents are working;if any microagent is down it should be restored immediately.It can further evolve factoring different constraints and conditions for more accurate estimation .

4 EXPERIMENTAL RESULTS

Our experimental setup contains a test gateway running on a single core,1GB RAM atom board running on Ubuntu 12.04,MQTT broker Mosquitto is installed on the gateway setting QoS 0 and retain flag as false.TCUP DM agent runs on atom gateway which communicates with three sensors namely temperature,humidity and light sensor over XBee, also an USB thermometer(RDing TEMPer) connected. An Arduino Uno is connected to gateway over serial port and a LED,as an actuator, is connected to GPIO pin of Arduino. Separate microagents are deployed on gateway for each individual sensor or actuator. Each microagent is written in Java using Paho MQTT client.DM server is hosted in same local network to eliminate network latency factor. DM service is running on a Spring4 Java Enterprise web application hosted on Apache Tomcat 8.24 and Oracle JDK7. Performance metric of DM service on Get/Put API is obtained running Apache Jmeter 2.13 and Perfmon plugin on a test client.The home gateway setup is depicted in figure 8.The hardware configuration of the test setup is shown in table 1.

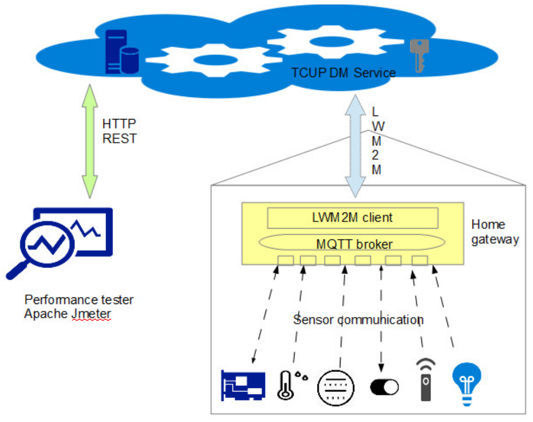

Figure 8: Home Gateway Setup.

Table 1: Hardware Configuration of Test setup

Infrastructure	Perf. test Client	DM Server
CPU Core	1	2
Memory(RAM)	2GB	4GB
CPU speed	2.4GHz	3.6GHz

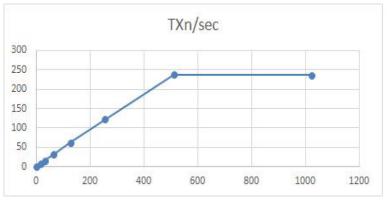

Figure 9: Query via microagents : Transaction rate vs Concurrent request.

4.1 Througput

Following section shows throughput(transaction per second) results for both query and update operation on aggregated sensor-block data. From figure 9 for Get Sensor data we see as concurrent request increases the througput initially increases but then it saturates beyond a threshold of concurrent requests. This is due to reaching the limit of computation capacity of the gateway. Similar is the behaviour in of sending instruction to actuators as depicted in figure 10. The througput is much lower than the read operation as write involves more latency in command confirmation from microagents.

4.2 Latency

Following section shows latency (response time) results for both query and update operation on aggregated sensor data in 90% cases. From figure 11 for Get Sensor data we see as concurrent request increases the latency doesn't change much initially but suddenly increases beyond a threshold. This is due to exhaustion of computation resource of the gateway. Similar is the behaviour for sending instruction to sensors as depicted in figure 12. The latency is much higher than the read operation as write involves more computation resource access. We can observe that GET API scales till 512 concurrent request whereas PUT call scales till 128 concurrent request.

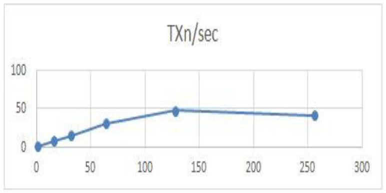

Figure 10: Send command via microagents : Transaction rate vs Concurrent request.

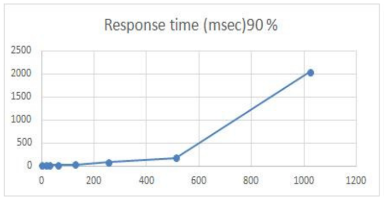

Figure 11: Query via microagents : Response time vs Concurrent request.

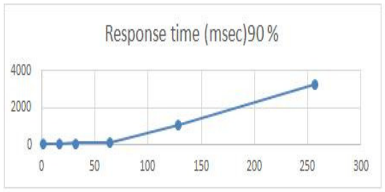

Figure 12: Send command via microagents : Response time vs Concurrent request.

5 CONCLUSIONS

In this paper we proposed a loosely coupled gateway agent where gateway agent communicates with end-devices through microagents. Gateway agents have LWM2M REST interface with MQTT pub/sub backbone underneath and MQTT based microagents support dynamic onboarding of end-devices without disrupting gateway agent operations. Security hardening of MQTT broker needs to be addressed during deployment. We also like to explore protocols like AMQP instead of MQTT and BSON instead of JSON to compare performance with this architecture.Implementation of event notification for sensor-block and automatic discovery of end-devices are our ongoing area of research.

ACKNOWLEDGMENTS

The authors would like to thank Prateep Misra, Soumitra Naskar and Arindam Halder for providing their valuable feedback and also thank Ritesh Ranjan for helping in performance test execution.

REFERENCES

[1] Open Mobile Alliance. 2011. OMA DM GwMO technical specification. http://www.openmobilealliance.org/release/GwMO/V1_1-20140617-C/OMA-TS-GwMO-V1_1-20140617-C.pdf. (2011).

[2] Open Mobile Alliance. 2017. OMA LWM2M technical specification. http://www.openmobilealliance.org/release/LightweightM2M/V1_0-20170208-A/OMA-TS-LightweightM2M-V1_0-20170208-A.pdf. (2017).

[3] Atilaneves. 2013. Go vs D vs Erlang vs C in real life: MQTT broker implementation shootout. https://atilanevesoncode.wordpress.com/2013/12/05/go-vs-d-vs-erlang-vs-c-in-real-life-mqtt-broker-implementation-shootout. (2013).

[4] L. Atzori, A. Iera, and G. Morabito. 2010. The Internet of Things: A Survey. *Computer Networks* 54, 15 (Oct. 2010), 2787–2805.

[5] D. Chattopadhyay and R. Dasgupta. 2012. A Novel Comprehensive Sensor Model for Cyber Physical System: Interoperability for Heterogeneous Sensor. In *6th International Conference on Sensing Technology (ICST)*. 179–183.

[6] D. Chattopadhyay, R. Dasgupta, R. Banerjee, and A. Chakroborty. 2012. Event Driven Video Surveillance System using City Cloud. In *47th Annual National Convention of Computer Society Of India (CSI 2012)*. Mcgrawhill.

[7] D. Chattopadhyay, R. Dasgupta, and A. Pal. 2013. Sensor Data Modeling for Smart Meters — A Methodology to Compare Different Systems. In *Proceedings of the 2013 International Conference on Computing, Networking and Communications (ICNC '13)*. IEEE Computer Society, 215–221.

[8] D. Chattopadhyay, A. Samantaray, and R. HariRaghav. [n. d.]. Lightweight Device Task Actuation Framework as IoT Test Platform. In *Internet of Things. IoT Infrastructures: Second International Summit, IoT 360 2015, October 27-29, 2015, Revised Selected Papers, Part II*. Springer International Publishing, 20–27.

[9] M. Collina, G. E. Corazza, and A. Vanelli-Coralli. 2012. Introducing the QEST broker: Scaling the IoT by bridging MQTT and REST. In *2012 IEEE 23rd International Symposium on Personal, Indoor and Mobile Radio Communications - (PIMRC)*. 36–41.

[10] ESR Consortium. 2017. LWM2M over MQTT profile specification. http://e-s-r.net/download/specification/ESR030-LWM2M-MQTT-1.0-A.pdf. (2017).

[11] S. Duttagupta, M. Kumar, R. Ranjan, and M. Nambiar. 2016. Performance Prediction of IoT Application: an Experimental Analysis. In *Proceedings of the 6th International Conference on the Internet of Things, IOT 2016, Stuttgart, Germany*. ACM, 43–51.

[12] Internet Engineering Task Force. 2014. Constrained Application Framework - CoAP. docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf. (2014).

[13] C. Gomez and J. Paradells. 2010. Wireless home automation networks: A survey of architectures and technologies. *IEEE Communications Magazine* 48 (2010), 92–101.

[14] IBM. 2013. Message Queuing Telemetry Transport - MQTT v3.1.1. https://www.iso.org/standard/69466.html. (2013).

[15] S. M. Kim, H. S. Choi, and W. S. Rhee. 2015. IoT home gateway for auto-configuration and management of MQTT devices. In *2015 IEEE Conference on Wireless Sensors (ICWiSe)*. 12–17.

[16] J.F. Kingman. 1960. The single server queue in heavy traffic. *Mathematical Proceedings of the Cambridge Philosophical Society* 57, 4 (1960), 902–904.

[17] T. Levä, O. Mazhelis, and H. Suomi. 2014. Comparing the cost-efficiency of CoAP and HTTP in Web of Things applications. *Decision Support Systems* 63 (2014), 23–38.

[18] J.D. Little. 1961. A Proof for the Queuing Formula. *Operations Research* 9, 3 (1961), 383–387.

[19] P. MISRA, A. Pal, C. BHAUMIK, D. KAR, S. NASKAR, S. ADAK, S. GHOSH, and et al. 2012. A computing platform for development and deployment of sensor data based applications and services. (2012). https://www.google.com/patents/WO2013072925A3?cl=en

[20] S. Nicholas. 2012. Power Profiling: HTTPS Long Polling vs. MQTT with SSL, on Android. http://stephendnicholas.com/posts/power-profiling-mqtt-vs-https. (2012).

[21] S. Oh, J.H. Kim, and G. Fox. 2010. Real-time Performance Analysis for Publish/Subscribe Systems. *Future Generation Computer Systems* 26, 3 (March 2010), 318–323.

[22] F. Renaud. 2016. Performance testing of serialization and deserialization of Java JSON libraries. https://github.com/fabienrenaud/java-json-benchmark. (2016).

[23] T. R. Sheltami, A.A. Al-Roubaiey, and A.S. Mahmoud. 2016. A Survey on Developing Publish/Subscribe Middleware over Wireless Sensor/Actuator Networks. *Wireless Networks* 22, 6 (Aug. 2016), 2049–2070.

[24] A. Sumaray and S. K. Makki. 2012. A Comparison of Data Serialization Formats for Optimal Efficiency on a Mobile Platform. In *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication (ICUIMC)*. ACM, 48:1–48:6.

[25] D. Thangavel, X. Ma, A. C. Valera, H.X. Tan, and C.K.Y. Tan. 2014. Performance evaluation of MQTT and CoAP via a common middleware. In *9th International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP),Singapore*. IEEE.

[26] VMWare. 2014. Choosing your messaging protocol AMQP, MQTT or STOMP. https://blogs.vmware.com/vfabric/2013/02/choosing-your-messaging-protocol-amqp-mqtt-or-stomp.html. (2014).

[27] R. Xia. 2015. Stress testing Mosquitto MQTT Broker. http://rexpie.github.io/2015/08/23/stress-testing-mosquitto.html. (2015).