Toward the Internet of Things Application and Management: A Practical Approach

Chen Zhou, Xiaoping Zhang
Department of Computer Science and Technology, Tsinghua University, Beijing, China
Email: c-zhou11@mails.tsinghua.edu.cn, zhxp@tsinghua.edu.cn

***Abstract*—The Internet of Things (IoT) has experienced a blooming development these years. Most of the current IoT applications are built in a highly vertical way, consisting of proprietary protocols and devices. This leads to the users bound in a single service provider, poor quality of service, redundancy of the core platform functionality and delay of the whole industry. IoT over IP brings the fundamental support for global communications as well as access to services and information. Several homogenous and seamless machine-to-machine (M2M) communication mechanisms have been proposed to extend the interoperability of smart objects. In this paper we present a unified design of message-centric architecture based on MQ Telemetry Transport (MQTT) for IoT application and management, and target on some practical problems and optimization for actual system.**

Keywords—Internet of things; interoperability; management; MQTT; pervasive communication.

I. INTRODUCTION

The Internet of Things aims to extend the Internet to ubiquitous goods in physical world, providing an intelligent environment by connecting things and people, which logically consists of three parts, smart objects, interoperability and applications.

Smart objects implies to daily things enabled with computing and communicating capabilities, normally embedded with a microcontroller and a low power radio. To interact with physical world, smart objects are often assigned with sensors and actuators, and a unique ID to be identified from others.

Smart objects are connected under different standards of physical layer, commonly WLAN, Bluetooth, 802.15.4, GPRS, or serial line etc. To achieve global interoperability, either complicated translation gateways are required, or unified IP network advocated by IPSO [1], between which the latter obtains growing supports in both academia and industry. 6LoWPAN [2] introduces an adaptation layer which enables IPv6 communication over 802.15.4; ZigBee IP provides end-to-end IPv6 networking and specifically supports ZigBee Smart Energy v2; There are also proposals of Low Energy Bluetooth IP solutions and so on.

Since the blueprint for Internet of Things has been proposed for years, many prototypes and products comes out with exciting ideas in areas of environment awareness and intervention, logistics, healthcare, and manufacturing etc. However, the deployment progress of all kinds of smart objects and applications is far from satisfying. Three main constraints restricting the development as we conclude:

- *Proprietary communication protocols.* Data aggregation and device controlling paradigms differ from vendor to vendor, which makes even straightforward need as turning off vendor P's light when persons leaving detected by vendor B's motion sensor hard to realize. Little secondary development can be done due to those closed solutions.
- *Security and privacy.* It's quite worrying that data collected from user daily life are processed and persisted in respectively services, with varying levels of security in transmission and storage. In consequence of limited computational capability, smart objects barely put strict authentication in practice [3][4].
- *Inconvenient to manage.* Currently no unified approach for smart objects management, which includes but not limited to initialization, configuration, upgrade and fault detection. Smart phone seems to bring possibility of interacting with all sorts of smart objects, while launching an app to turn off a light, and another for a fan has nothing to do with convenience.

MQ Telemetry Transport (MQTT) [5] is a simple light-weight machine-to-machine protocol designed for constrained devices and unreliable networks. It has a 2 bytes fixed-length header, one-to-many message distribution and 3 QoS levels for message delivery. Additional brilliant built-in properties like clean-session, retain-message as well as will-message make it more suitable for most of the Internet of Things scenarios.

In our point of view, a prospective future of the Internet of things requires a unified human machine interface, and an agreed mechanism for information interaction among all smart objects is the heading direction. In the rest of this paper we first present some related work of emerging protocols and paradigms for IoT application level interoperability in Section II and qualitative analysis on the strengths and weaknesses; Then we propose a practical message-centric architecture based on MQTT for IoT application and management in Section III; In Section IV we address on some realistic problems and publisher predominant MQTT protocol extensions when we put the design scheme into realization; And we present Wi-Fi enabled prototypes for mechanism verification in Section V; Finally the Conclusion is given in section VI.

II. RELATED WORK

A. Resource-Oriented Paradigm

Referred to the well-known HTTP in the Internet, Constrained Application Protocol (CoAP) [6] is a realization of Representational State Transfer (REST) architecture for

constrained nodes and networks, where all functions are encapsulated as addressable resources and achieved via methods of GET, PUT, POST, and DELETE. CoAP is based on request/response interaction model, in which clients initiate requests for some resources identified by URI, servers process the requests and return appropriate responses around the transfer of representations. The active IETF draft also defines an observation mechanism for CoAP, that is to say, one request for certain resource, and continuous responses from server.

Although lots of knowledge, successful experience and tools from HTTP can be easily integrated to CoAP, the defects of resource-oriented paradigm are as obvious.

- Requests, responses and observations are all loaded on smart objects, while power constrained devices can hardly afford, much less when high concurrency.
- Duplicated requests and observations for same single resource cost a waste of computing and networking.
- IP address of each device must be known, so the service discovery mechanisms like SLP, mDNS are introduced, complicating the structure.
- Network address translation (NAT) issue faced for global accessibility in currently most common IPv4 subnet environment like home WiFi scenario.

B. *Message-Oriented Paradigm*

Classified by message distribution mode, two of the most common message-oriented architectures are message queuing and publish/subscribe messaging [11]. In message queuing, also called point-to-point, each message has to be assigned with a definite receiver, and only one receiver obtains the message. While in pub/sub messaging, subscribers specify their interest in messages of certain type or topic, and receive messages asynchronously once a publisher publishes a message on the registered interest. Come to say in the IoT scenario, the major competitive edge of pub/sub messaging is the loose coupling between business logic and smart devices, and bringing many-to-many communication availability.

Under message-oriented paradigm, many protocols have been proposed and widely applied in typical areas: Extensible Messaging and Presence Protocol (XMPP) for instant messaging tools like Gtalk; Simple Sensor Interface protocol (SSI) for communication between user terminal and smart sensors from Nokia; MQTT for smart phone notification push service, for example Facebook Messenger; Advanced Message Queuing Protocol (AMQP) for enterprise datacenters like JP Morgan; Streaming Text Oriented Messaging Protocol (STOMP) for only text based human computer communication; WebSocket for full-duplex communication between browser and web server.

Integration and extension works among these public message protocols are underway in recent years to fit the IoT world, while MQTT is outstanding for its simplicity and efficiency, which costs only a small footprint and low power consumption on embedded devices, meanwhile guarantees the reliability and flexibility for message distribution. In [14] they explore the application of the publish/subscribe messaging paradigm and propose a programming model to environmental, monitoring, and control systems. [13] presents the packet loss and delay of different MQTT QoS levels under realistic network environment. A hot-topic based algorithm and correlated MQTT extension are proposed in [12]. MQTT-SN (formerly MQTT-S), is a variation of MQTT based on non-TCP/IP networks. In [9] presents the performance evaluation of MQTT-SN and CoAP. Features and usability of MQTT and AMQP are compared in [10]. The comparison between CoAP and MQTT has been addressed in [7][8], respectively considering the cases the two protocols running on power constrained gateways and smartphones.

III. SYSTEM DESIGN

A MQTT-based IoT architecture for publish/subscribe message-oriented paradigm is straightforward as shown in Fig.1. Smart devices, applications and the manager setup only connection to brokers service and communicate through message publication and subscription on specific message topics. Devices, apps and the manager are all considered as standalone MQTT clients, while brokers service might be implemented as single host or federated to improve scalability and reliability.

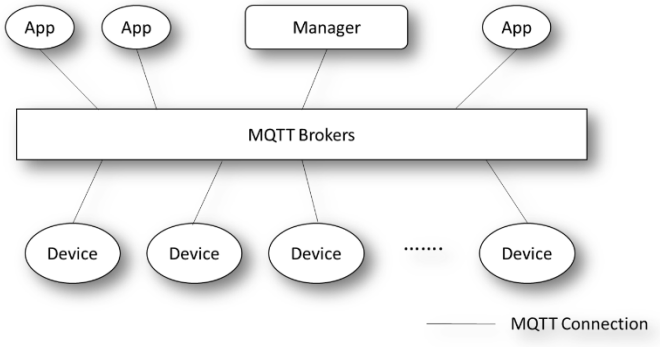

Fig. 1. MQTT-based IoT Architecture

In the aspect of application, smart devices often play a role of sensor, actuator or combination of both, which respectively means to publish sensed data on report topic, to subscribe on manage topic for action commands or the union. All business logic like periodical operation or conditional interacting between devices are taken care by different independent apps, decoupling devices and applications and simplifying the development and maintenance.

In the aspect of management, a suite of device initializing, recognizing and registration; network connection and identity assignment; configuration; monitoring and fault detection; on-the-air app and firmware update and factory reset are considered necessary but not sufficient. To apply the management scheme, three participants of smart devices, brokers service and the manager need to collaborate and work together. On the brokers service side, there are emerging open source software implements like Mosquitto or hardware implementations like IBM MessageSight. Brokers service is some degree of scalable thanks to the ability of bridging.

In the rest of this section we mainly focus on the device side and manager side, presenting a practical framework for IoT application and management.

A. *Smart Device State Machine*

Smart devices are always within one of the four states: initializing, running, offline and reconnecting as shown in Fig.2.

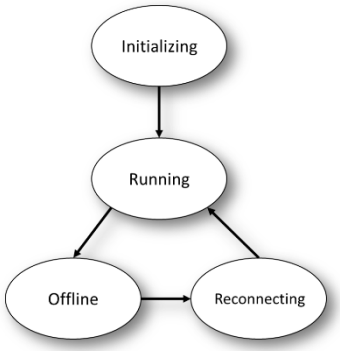

Fig. 2. Smart Device State Machine

1) Initializing

First of all, a fresh-from-factory smart device has no conception of the end-user specific network, for example home WiFi SSID and password, demanding an agreed mechanism to setup these information for device Internet connection which is discussed in Section IV. After the message channel established, the device has to declare what itself is for manager reorganization and registration. Finally parameters for device operating, especially topics to report data and receive commands, have to be set during the initializing stage.

Each networkable device has a universally unique 48-bit MAC address, which is used as the identifier UUID in the initializing stage. A self-describing message containing all product information including manufacturer, model, item number, firmware version and loaded apps etc., is published on the INCOMING_DEVICE/<UUID> topic, acting as the device registration. Meanwhile, the device subscribes on the topic DEVICE/<UUID>, waiting for initialization from manager service. Normally device receives a configuring message with default settings and a unique client identifier if the device is recognized and verified by the manager service. And the device report topic and manage topic will be reset, having default INCOMING_DEVICE and DEVICE topic prefix abandoned after the initialization stage, for device privacy and easy permission control. Finally a reconnecting operated to bring changes into effect. Fig.3. shows the workflow of a new coming device initializing.

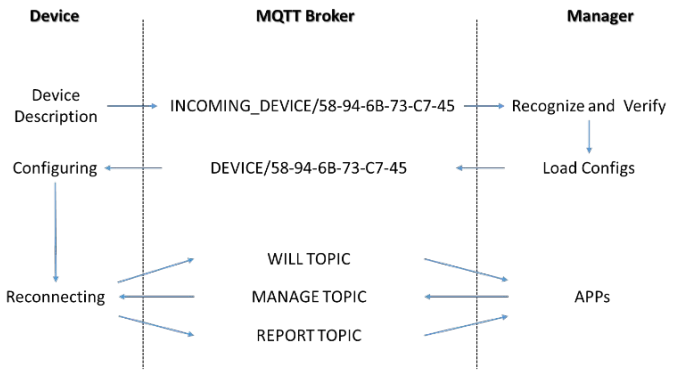

Fig. 3. Device Initializing Workflow

2) Running

Once the initialization is done, the device comes to the running stage, conducting functionalities varies from different types of devices. For example, a smoke detector senses the air and reports to certain topic, while a light subscribes on some topic for on/off messages to react.

All devices with a manager agent subscribes on the given manage topic, waiting for manager commands including reconnect, reboot, configure, upgrade, report, reset and so on, and reporting device status like power-remained, running app information and unexpected exceptions, actively or passively. The management protocol could be updated together with the upgrade of firmware to support extending management schemes.

3) Offline

Under two circumstances might a device switch to offline state. Regular offline is a mechanism for device energy saving, which goes sleep and wakes up performing alternately. Device has to publish offline message for manager awareness in regular offline, and save context for later recovery. Irregular offline happens when network failure, power exhausted or even device damage, on which user interference is often needed to recover the device functionality. Irregular offline will be detected by manager service by a will message delivered, or no heart beat report for a long timeout, and then an alert message raised to keep relevant people informed.

4) Reconnecting

Reconnecting message will be published after device regular offline. For irregular offline, in the case which device remains in the status before passing out, just reconnects and relieves the alarm. While if the device lost all context and reset to the initializing state, which publishes self-description information to the INCOMING_DEVICE/<UUID> topic acting as a fresh-from-factory, manager service will quickly restore the saved settings to the device and cleanup the context for the last connection. After reconnecting done, device comes to running stage again.

B. Framework of Smart Device

The framework of smart device is layered into two, the core (or called firmware) and the applications, as shown in Fig.4. The core layer takes responsibility of initialization, configuration, management, and application loading and scheduling. Messages communication is also processed in the core layer. Messages received from MQTT brokers are queued in massage handler, and then switch to corresponding applications or the manager agent according to the message topic. The applications layer contains apps which are just implementations of various business logic under a defined standard structure named app interface. There could be several different apps running on same device, while all of them are under the management of single core process. Fig.5. presents the workflow of core process.

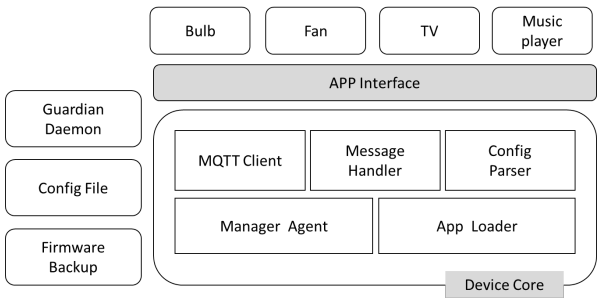

Fig. 4. Framework of Smart Device

All the specific parameters the device needs in booting, loading, connecting and running are organized into key value pairs in one config file, including four sections of device properties, messaging properties, application properties and manager properties. These settings can be read and modified though network configuration. Some additional pairs can be added ahead of application or firmware upgrade. So once the upgrade done, all settings stand in place.

1) Application Interface

App interface is the regulation for application layer development. Each app should have the functions declared in the interface implemented to make device work properly. All configurations an app need to run will be actively set by core process after loaded. And the message communication methods for apps to call, as publish, subscribe and unsubscribe, are provided at the same time.

on_connected: Called once MQTT connection with broker established and device registration done. App initializes pub/sub related business in this function.

on_shutdown: Called when device going to shut down or unload certain app. App saves context for later recovery, and cleans up resources and terminates any sub process forked from it, for example standalone media player.

main_loop: Called in every main loop of core process. Application major logic is implemented here. No block or long holding the process, but return or yield in short piece of time, leaving unfinished work continued on next round.

on_message_received: Called when message received for app, usually some commands that app subscribes. App then performs the corresponding action.

update_configs: Called when loaded or configs modified from manager, since configs are managed by core process and app only gets access to specified section.

2) Manager Agent

The manager agent now performs actions of initializing, configuring, fault reporting, apps and firmware upgrading, factory reset, reconnecting, rebooting and power management.

Commonly smart objects are manufactured in batch, so that single device knows nothing about end user specific settings. Initializing is to register device in manager service, bind to corresponding user, and configure as user favor. All configurations from manager are persisted in the device config file for later runs.

For apps and firmware upgrade, the manager agent parses updated source code from manager messages, and replaces the running one on flash. When a running app is upgraded, the manager agent just reloads the app module; while for the firmware, the manager agent has to terminate by purpose, letting the guardian daemon to restart it over.

Reconnecting and rebooting is useful when migrating lots of devices from one broker to another, or after agreed message topics or will note modified. When factory reset triggered, the manager agent cleans the running workspace and extracts factory backup from archived package to overwrite the current one, including core code, apps code, configurations and resources, and finally reboots to take effects.

The device status like connected, disconnected and expired is under monitor of manager service, together with device description and running apps information. While hardware module support is required for reading remained power of a battery restrained device.

3) Guardian Deamon

Guardian daemon is a standalone service, which takes responsibility of starting the core process when device powered up, keeping the PID and waiting for core process to crash or exit as purpose. Then the guardian daemon restarts the core process to make the device continuous operating. Commonly device core process exits under three circumstances, unexpected exception occurs so that process crashes down, management command of rebooting received, or firmware upgrade done to restart new core. The guardian daemon distinguishes all the cases by retrieving the exit code, and acts correspondingly.

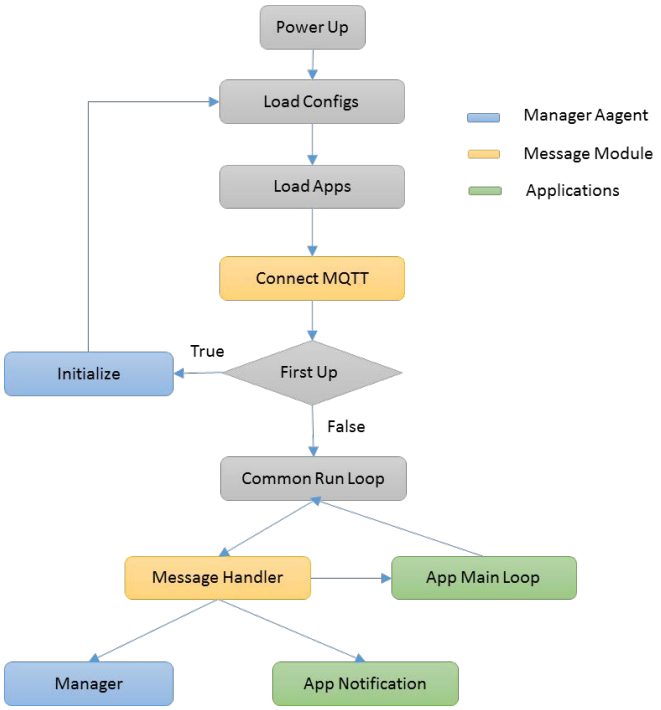

Fig. 5. Smart Device Core Process

C. Manager Service Framework

Mapping to the manager agent on smart device, the manager service from the cooperation on the other side implements management functionalities of device initialization, registration, configuration, monitoring, and other administrator commands etc. The management scheme may be performed automatically or manually.

Fig.6. shows the framework of manager service. From top to bottom, the manager service provides responsive web interface along with related web services, displaying status and detailed information about the coming smart devices, and sending management messages to devices according to network administrator or device owner operations. Persistence layer stores all registered and incoming devices objects, the manager configurations and running logs. Then comes the concrete management business layer described below. Finally the communication layer which contains an MQTT client as publish/subscribe communication agent.

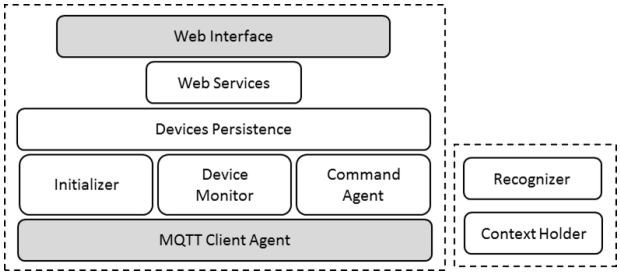

Fig. 6. Manager Service Framework

The business layer consists of three main components. Initializer handles device initializing messages, recognizes device model, registers and configures devices at the first time. Device monitor keeps monitoring connected devices on the fields including basic functionalities, network conditions, and power remained, recording and reporting unexpected exceptions. Command agent interprets and forwards user commands posted from web interface to certain device manage topic, dealing with all manual or scriptable management jobs.

Take on-the-air app upgrade, one of the management features, as example. Although this could usually be done automatically, we manually start the workflow from the very beginning as shown in Fig.7. User uploads updated app source file to the web service though an HTTP post request. Standard web container extracts the file object from the multipart request, and publishes it as an MQTT message payload encoded in UTF-8 to certain device manage topic, with the agreed management message format of upgrading application. Finally the manager agent on smart device performs an app upgrade operation as described above.

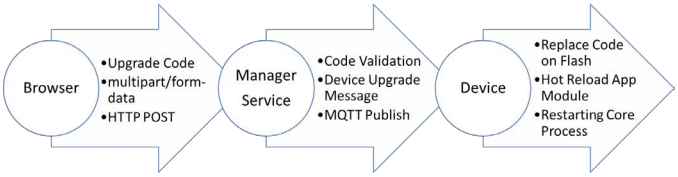

Fig. 7. On-the-air Upgrade Workflow

IV. DISCUSSION

A. User Specific Network Access

The first step for a fresh-from-factory IoT smart device to start working is to obtain the access to the Internet in user specific environment. Considering WiFi as the most common situation, for devices like smart phone or tablet, the habitual way is that user chooses a wireless network among all available ones listed on the screen, and types password in if required. However for the most other smart objects which barely have user I/O interface, an convenient technique to configure information of user network is quite necessary. Fortunately the everywhere smart phones brings solutions.

One straightforward idea is that the smart object starts a transient WiFi AP, and user smart phone joins in it then sends the realistic network information. However this limits to only configuring single device at one time.

To improve the parallel configuring, a feasible idea is that smart phone keeps broadcasting mock beacon frames (or probe request), packaging real network information in the field of SSID, while smart objects capture the frame and extract it. For example the real user network gets SSID "OpenWrt" and password "Fit1-216", the mock AP the smart phone pretends would be like "OpenWrt#@@#Fit1-216", in a format which separates SSID and password explicitly. This idea is easy to implement but gets a constraint of the mock SSID length, 32 octets maximum.

A more complicated idea is that the smart phone encodes the network information into a series of special frames filled by meaningless bytes, but makes the frame size significant, for example a frame with size 597 stands for the character "M". and sends the frames in an predefined pattern cyclically. While the smart objects monitor all the traffic and find out the frame size pattern from fixed source to destiny. To the best of our knowledge, TI implements a proprietary mechanism of this idea in their latest WiFi chip CC3000 called Smart Config.

B. Publisher Predominant MQTT Extensions

1) Publication Wildcard

As defined in the MQTT specification, two wildcard characters are supported for subscription: A '#' represents a complete sub-tree of the topic hierarchy, such as SENSOR/#; A '+' represents a single level of the topic hierarchy, such as SENSOR/+/TEMP. This wildcard feature makes it simple for subscribing polymerized topics, for example reading all temperature sensors in a building within one subscribe message. On the contrary, no wildcard for publication is supported in the current version, which we find quite necessary when put the management scheme in batch. So is for intervention category applications such as turning off all the lights, which we need is to publish an "off" message to HOME/LIGHT/#, while keeping the capability of controlling respectively.

2) Delayed Confirm Message

MQTT defines three levels of Quality of Service (QoS), QoS0 for at most once; QoS1 for at least once; and QoS2 for exactly once. The QoS aims to ensure message delivery, but no care for the consequence of the message. For example an "off" message published with QoS≥1 only indicates that the light receives, but no guaranty of the light's really off. To achieve this, another topic is required for the light to publish its status, while complexity introduced especially for complicated applications.

We define the delayed confirm message, identified by the reserved QoS3. Similar to QoS2, the delayed confirm message gets 4 rounds as PUBLISH, PUBREC, PUBREL and PUBCOMP. The difference for subscriber with QoS3 is that not responses the PUBCOMP until the message semantically performed. While for the publisher, PUBCOMP from broker only arrives when all connected QoS3 subscribers response accomplished. Instead of a fixed retry timeout, the delayed confirm message timeout is determined by subscribers while publishers get informed in PUBREC.

With the help of delayed confirm message, we can even build point to point communication upon the publish/subscribe structure, by attaching response messages on PUBCOMP payload, although which does not fit the philosophy of MQTT.

V. PROTOTYPE VERIFICATION

To verify the whole architecture and workflow of the pub/sub message-oriented paradigm as proposed above for IoT application and management, we build several smart object

prototypes including a “smart” bulb, a fan, an Internet music player and a television as shown in Fig.8. All these prototypes are based on Raspberry Pi, a credit-card-sized embedded development board equipped with a WLAN card. The device core process is the same while apps differs, all of them written in python. The bulb app is straightforward which subscribes for controlling commands and turns high/low of the on-board General Purpose I/O (GPIO) output signal pin accordingly. The fan app is almost the same but an external amplifying circuit is required for the insufficiency of GPIO output power. The music player retrieves user specified song from online music library by the name from certain topic. While for the television, available channels are published as retain message on report topic, and switches according to user messages.

Fig. 8. Prototypes of Smart Objects

The manager service is a java implemented web application which can be driven in any web containers. The manager service has configurable rules set for device management automation, meanwhile keeps the ability of manual operation through web services. Persistent devices list and detailed information of each device are displayed on the manager webpage, together with user interfaces for device configuration, upgrade, and other commands of reconnect, reboot, factory reset etc. Screenshot of the manager web is shown in Fig.9.

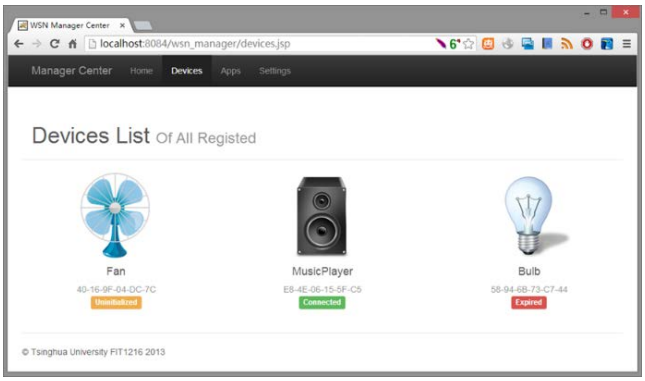

Fig. 9. Manager Web Screenshot

As for the MQTT communication part, we use the open source Mosquitto broker. Integrating with the python and java clients from Eclipse Paho project respectively runs on the device and the manager service.

VI. Conclusion

The Internet of Things has experienced a rapid development in these years. However, the majority of IoT applications nowadays, are built in extremely vertical way. That is to say, manufacturers and providers using proprietary protocols and devices to realize certain fixed application. This not only puts restrictions on the interoperability between smart objects, but also exposes risks of security and privacy by those jagged mechanisms for user data transmission and storage. On the other hand, those vertical applications make it scarcely possible to realize unified human-objects interface, bringing inconvenience for IoT application and management.

In this paper we compare two kinds of information exchange paradigms, as resource-oriented and message-oriented, while the latter one suits better to smart objects with restricted computing, storage, networking and energy resources. We propose a practical system design for MQTT based IoT application and management framework. We then discuss on realistic problems of user specific network access, and come up with MQTT protocol extensions for better IoT scenario adaptation. Finally we present the prototype implementations of smart objects and the manager service.

Acknowledgment

This work is supported by National High Technology Research and Development Program of China (863 Program) under Grant No. 2011AA010704.

References

[1] A. Dunkels and J.-P. Vasseur, “IP for Smart Objects,” IPSO White Paper, Sept. 2008.
[2] Mulligan, Geoff. "The 6LoWPAN architecture." Proceedings of the 4th workshop on Embedded networked sensors. ACM, 2007.
[3] Atzori, Luigi, Antonio Iera, and Giacomo Morabito. "The internet of things: A survey." Computer networks 54.15 (2010): 2787-2805.
[4] Heer, Tobias, et al. "Security Challenges in the IP-based Internet of Things." Wireless Personal Communications 61.3 (2011): 527-542.
[5] “MQTT v3.1 specification” available online: http://mqtt.org/.
[6] “Constrained Application Protocol (CoAP)”, draft-ietf-core-coap-18 available online: https://datatracker.ietf.org/doc/draft-ietf-core-coap/
[7] Bandyopadhyay, Soma, and Abhijan Bhattacharyya. "Lightweight Internet protocols for web enablement of sensors using constrained gateway devices." Computing, Networking and Communications (ICNC), 2013 International Conference on. IEEE, 2013.
[8] Niccolò De Caro, Walter Colitti, Kris Steenhaut, Giuseppe Mangino, Gianluca Reali. “Comparison of two lightweight protocols for smartphone-based sensing.” Communications and Vehicular Technology (SCVT), 20th Symposium on. IEEE, 2013
[9] Davis, Ernesto García, Anna Calveras, and Ilker Demirkol. "Improving packet delivery performance of publish/subscribe protocols in wireless sensor networks." Sensors 13.1 (2013): 648-680.
[10] StormMQ, “A Comparison of AMQP and MQTT”, White Paper.
[11] Kim, Minkyong, et al. "Efficacy of techniques for responsiveness in a wide-area publish/subscribe system." Proceedings of the 11th International Middleware Conference Industrial track. ACM, 2010.
[12] Domínguez, Augusto Morales, et al. "A Hot-topic based Distribution and Notification of Events in Pub/Sub Mobile Brokers." Network Protocols & Algorithms 5.1 (2013).
[13] Lee, Shinho, et al. "Correlation analysis of MQTT loss and delay according to QoS level." Information Networking (ICOIN), 2013 International Conference on. IEEE, 2013.
[14] Stanford-Clark, Andy J., and Glenn R. Wightwick. "The application of publish/subscribe messaging to environmental, monitoring, and control systems." IBM Journal of Research and Development 54.4 (2010): 1-7.